Comenius University
Faculty of Mathematics, Physics and Informatics

Surface Triangulations Dependent on Geometric Data

Project of Dissertation

Study program: Geometry and Topology
Department: Department of Algebra and Geometry
Supervisor: doc. RNDr. Andrej Ferko, PhD.

Bratislava 2023 Ing. Martin Čavarga

Abstract

The generic irregularity of mesh representation allows for a wide variety of configurations, enabling the development of various algorithms for their processing. Our focus is on triangle meshes that tessellate surfaces representing boundaries of solid objects, covering methods for extracting volumetric data from meshes and using this spatial data to reconstruct optimized meshes from them. The work also includes the stabilization of the Lagrangian shrink-wrapping method for wrapping target mesh geometries with arbitrary topology and the use of progressive meshes for mesh simplification, view-dependent mesh optimization, and mesh compression used in state-of-the-art methods for geometry optimization, for example, in modern game engines.

Contents

List of Figures

Introduction

In the realm of 3D data storage and display, polygonal meshes are a widely adopted representation. Meshes are, of course, not the only way of representing spatial data, but perhaps their most alluring aspect is that despite containing a finite amount of information, the irregularity of their generic forms allows for a wide variety of configurations, and thus requires all kinds of algorithms utilized for their processing. In this thesis, we orient ourselves on the mesh representations, particularly triangle meshes tessellating surfaces representing boundaries of solid objects. In particular, we dive deeper into methods of extracting volumetric data from meshes, as well as using this spatial data to reconstruct optimized meshes from them once again.

We begin with a theoretical chapter on three main types of geometric representations: *meshes*, *functional representations*, *voxel fields*. This chapter introduces a deeper mathematical language with focus on mesh representations, and their role in conversion to or from image-based or fully-implicit data. We also include some practical results. Furthermore, we prove a result for estimating the number of mesh primitives after subdivision which is then used as a foundation for stabilizing the numerical method used in the following chapter.

The next chapter discusses Lagrangian shrink-wrapping and its practical implementation for wrapping target mesh geometries. This method can also be extended to point cloud target data. Not only do we stabilize the method for shrink-wrapped target meshes with arbitrary topology, but also allow for using a starting surface from an isosurface of the signed distance field to the target surface.

We follow by providing an overview of mesh simplification techniques using progressive meshes. We discuss how these techniques are used for view-dependent mesh optimization and mesh compression. Additionally, we mention the current state-of-the-art methods of geometry optimization for modern game engines.

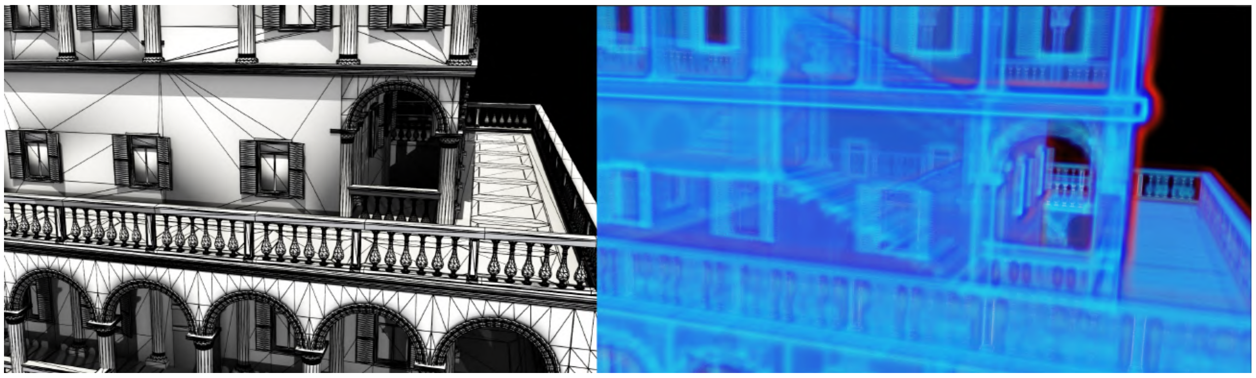


Figure 1: Triangle mesh (left), and volumetric (right) renderings of the same scene [75].

Chapter 1

Representations of Geometric Data

Because of their relatively low memory requirements, point cloud and surface representations of geometric objects comprised the majority of 3D models across virtual environments throughout history. With the advancements in data storage and processing speed, other general forms became viable, such as voxel or fully functional representations. In the domain of engineering applications, such as *Finite Element Method* (FEM), there is demand for discrete versions of real-world objects where values of physical quantities are computed primarily at *node (vertex) elements*, and secondarily at *edges*, *faces* or *volumes*. Purely visual applications, on the other hand, require only data required for rendering the objects as shapes or intensity values on the screen.

In this chapter, we formulate mathematical descriptions of geometric representations and processing / conversion operations on them, essential for the main results of this work. Although the range of possible representations in various data structures is, of course, much wider than the chosen mathematical description, formal definition becomes crucial when securing correct conversion among different representations, namely because of compatibility between frameworks as well as changing requirements for memory consumption, access and processing speeds.

For all intents and purposes of this work, geometric objects are defined in ambient n-dimensional Euclidean space $\mathbb{E}^n$, with $n = 3$ for most cases. This implies the use of real-valued vectors from vector space $\mathbb{R}^n$ equipped with standard *dot product* $\cdot$ or $\langle\cdot,\cdot\rangle$, and also for $n = 3$ the *cross product* $\times$ between them. The aforementioned spaces are isomorphic, therefore we distinguish between vectors $\mathbf{v}_{\mathbb{R}^n}$ (defined at the origin $\mathbf{0} \in \mathbb{R}^n$) and their analogues defined at points $\mathbf{p}_{\mathbb{E}^n} \in \mathbb{E}^n$ only by being able to apply inner and cross product on the former on top of standard addition and scalar multiplication.

Naturally, the dot product induces the *standard Euclidean norm* $\|\cdot\|$, and when combined with the cross product $\|\cdot \times \cdot\|$ also an *area form*, or alternatively a *volume form* $\langle \cdot, \cdot \times \cdot \rangle$. Vectors $\mathbf{v} \in \mathbb{R}^n$ are, of course, computed from points $\mathbf{p} \in \mathbb{E}^n$, for example surface normals, velocities, acceleration, field gradients etc. One can easily proceed to define tensors at points $\mathbf{p} \in \mathbb{E}^n$ in a similar fashion.

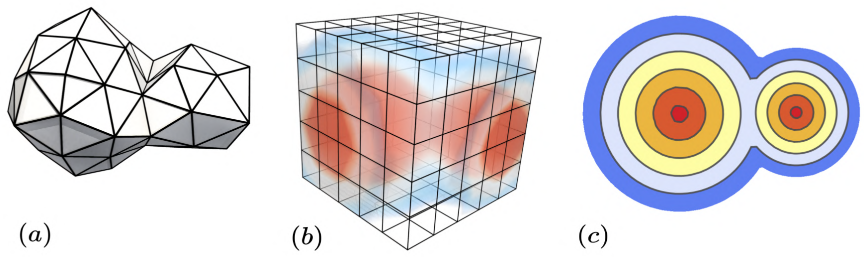


Figure 1.1: Three key types of geometry representations in $\mathbb{E}^3$ used in this thesis: (a) mesh, color plots of (b) scalar values sampled at a voxel field, and (c) a functional representation (FRep).

1.1 Mesh Geometry

In general, we distinguish between two key approaches to defining the same type of entity - a *polyhedral mesh*:

- *combinatorial* - description in terms of *vertex identifiers*: $V_{comb} = \{1, ..., N_V\}$ (indices from $\mathbb{N}$ or $\mathbb{Z}$ or any other identifying type) accompanied by *connectivity information*, that is: a subset of the power set $2^{V_{comb}}$,
- *geometric* - mapping vertex identifiers V_{comb} to ambient space as a *point cloud*: $V = \{\mathbf{v}_1, ..., \mathbf{v}_{N_V}\} \subset \mathbb{E}^n$, and handling their convex hulls (edges, faces, volumes).

It is often important to maintain the distinction between connectivity via vertex identifiers, and the corresponding geometry in ambient space. The former (combinatorial) approach will be important for the definition of *tessellation-changing operations* such as *flip*, *split*, and *collapse*. These produce new combinatorial structures from given input. The geometric approach becomes crucial when simulating processes on the mesh geometry in space, for example, in *Laplacian fairing* (see Chapter 4 in [11]) which evaluates a *mesh Laplacian* for each vertex $\mathbf{v}_i \in \mathbb{E}^n, i = 1, ..., N_V$.

An experienced reader realizes that the combinatorial structure allows for a wide variety of potentially *degenerate configurations*. The geometric approach, on the other hand, generally places a set of restrictions on the representation itself. In practice, meshes have a substantial amount of degeneracies which need to be repaired, in order to be prepared for geometric algorithms which accept "well-behaved" surfaces or tessellated volume regions as input.

1.1.1 Definitions of a Mesh

We start with the more straightforward combinatorial approach using finite sets of elements extended to collections of their subsets:

Definition 1.1.1. (Abstract Simplicial Complex) Let $V_{comb} = \{1, ..., N_V\}$ be the set of *vertex identifiers*. A set $K = \{i_1^K, ..., i_k^K\} \subset V$ is referred to as an *abstract* k*-simplex*. A collection $\mathcal{K} \subseteq 2^{V_{comb}}$ is called an *abstract simplicial complex*, if for every k-simplex $K = \{i_1^K, ..., i_k^K\} \in \mathcal{K}$, every $(k-l)$-simplex $E \subseteq K$ is also contained in $\mathcal{K}$ for $l = 0, 1, ..., k$. We say that $\mathrm{Cl}(\mathcal{S}) = \{E \in \mathcal{K} : E \subseteq K \in \mathcal{K}\}$ is a *closure* of $\mathcal{S} \subset \mathcal{K}$. A *star* of a k-simplex $K \in \mathcal{K}$ is defined as $\mathrm{St}(K) = \{L \in \mathcal{K} : K \subset L\}$. The set difference $\mathrm{Cl}(\mathrm{St}(\mathcal{S})) \setminus \mathrm{St}(\mathrm{Cl}(\mathcal{S})) = \mathrm{Lk}(\mathcal{S})$ is referred to as the *link* of $\mathcal{S} \subset \mathcal{K}$.

The abstract representation of $\mathcal{K}$ may represent a wide range of finite datasets with some inherent connectivity. As such they are not geometric, but can be mapped onto points in space:

Definition 1.1.2. (Topological and Geometric Realizations) Let $\mathcal{K}$ be an abstract simplicial complex with vertices $V_{comb} = \{1, ..., N_V\}$. Map $|\cdot| : \mathcal{K} \to \mathbb{R}^{N_V}$ such that $|i| = \hat{\mathbf{e}}_i, i = 1, ..., N_V$ is called a *topological realization of* $\mathcal{K}$. Let $\Phi_V : \mathbb{R}^{N_V} \to \mathbb{E}^n, n \in \mathbb{N}$ such that $\Phi(\hat{\mathbf{e}}_i) = \mathbf{v}_i \in V = \{\mathbf{v}_1, ..., \mathbf{v}_{N_V}\} \subset \mathbb{E}^n$. Image $\Phi_V(|\mathcal{K}|)$ is referred to as *geometric realization of* $\mathcal{K}$.

Hoppe et al. [32], define a triangle mesh as a pair $(\mathcal{K}, V)$ where the abstract simplicial complex $\mathcal{K}$ carries combinatorial and vertex set $V \subset \mathbb{E}^3$, on the other hand, describes geometry. Additionally, the remaining bridge towards surfaces or volume regions in Euclidean space which meshes are expected to discretize, needs to be built using convex combinations. Hence, we delay the complete definition of a mesh until we describe it from the geometric point of view, as what can essentially be summarized as: linear shapes glued together.

Definition 1.1.3. (Polytope, Simplex) Let $\{\mathbf{x}_1, ..., \mathbf{x}_N\}$ be a finite set of points in $\mathbb{E}^n$. We say that the convex hull $\mathrm{conv}(\{\mathbf{x}_1, ..., \mathbf{x}_N\})$ is a *polytope*, and for $N = k$ affinely independent points in $\mathbb{E}^n, k \leq n$ we refer to such polytope as a k*-simplex*.

k-simplices are k-dimensional generalizations of line segments, triangles and tetrahedra. They are the simplest shape that can exist as a set of k-dimensional *convex combinations* of independent points

$$\sum_{i=1}^{k} \alpha_i \mathbf{x}_i \ , \quad \sum_{i=1}^{k} \alpha_i = 1 \ , \alpha_i \geq 0, i = 1, ..., k.$$

Every k-simplex is also a k-polytope. As we connect simplices into a larger structure, we need a suitable tool to express the dimensionality of their geometry.

Definition 1.1.4. (Hyperplane) Let $\Pi = \{\mathbf{x} \in \mathbb{E}^n \ : \ (\mathbf{x} - \mathbf{x}_0) \cdot \mathbf{n} = 0\}$ for some $\mathbf{x}_0 \in \mathbb{E}^n$ and *normal vector* $\mathbf{n} \in \mathbb{R}^n$ be a *hyperplane*, and $\Pi^+ = \{\mathbf{x} \in \mathbb{E}^n \ : \ (\mathbf{x} - \mathbf{x}_0) \cdot \mathbf{n} > 0\}$, $\Pi^- = \{\mathbf{x} \in \mathbb{E}^n \ : \ (\mathbf{x} - \mathbf{x}_0) \cdot \mathbf{n} < 0\}$ the open *half-spaces* bounded by hyperplane Π. Furthermore, we denote *closed half-spaces* as $\overline{\Pi^+} = \Pi \cup \Pi^+$ and $\overline{\Pi^-} = \Pi \cup \Pi^-$.

Definition 1.1.5. (Supporting Hyperplane, Face) A hyperplane Π is said to be a *supporting hyperplane* to a k-dimensional polytope K if $\Pi \cap K \neq \varnothing$ and $K \subset \overline{\Pi^+}$ or $K \subset \overline{\Pi^-}$. The non-empty intersection $F = \Pi \cap K$ is referred to as a $(k-l)$-*face* (for $l = 0, 1, ..., k$) of K.

For a 3-dimensional polytope in $\mathbb{E}^3$, 2-faces are the 2-polytope walls, 1-faces are edge line segments, and 0-faces are individual vertices. Of course, by definition there are also two *improper faces*, namely the 3-polytope itself and $\varnothing$. By Definition 1.1.5, we cover the entire hierarchy of points $\mathbf{x} \in K$. From now on, we denote $k \leq n$ as the maximum dimension of a face of a given polytope unless we say otherwise to declutter notation.

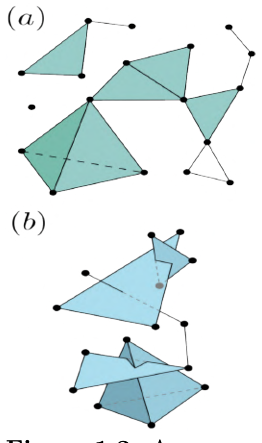


Figure 1.2: An example (a) and a counter-example (b) of a *simplical complex.* Source: Wikimedia Commons.

Definition 1.1.6. (Simplicial Complex) A *simplicial complex* is a collection $\mathcal{K}$ of simplices satisfying:

(1) Every $(k-l)$-face of $\mathcal{K}$ is also contained in $\mathcal{K}$ for $l = 0, 1, ..., k$.

(2) If $K_1 \cap K_2 \neq \varnothing$ for $K_1, K_2 \in \mathcal{K}$ then $K_1 \cap K_2$ is a face of both K_1 and K_2.

Let $\mathrm{St}(\{\mathbf{v}\})$ be a star of vertex $\mathbf{v} \in \mathbb{E}^n$, then $\mathcal{N}(\mathbf{v}) = \{\{\mathbf{w}\} \in \mathcal{K} : \{\mathbf{w}\} \in \partial\mathrm{St}(\{\mathbf{v}\})$ is a vertex in $\mathcal{K}\}$ is referred to as the *neighborhood* of vertex $\mathbf{v}$. Furthermore, for simplex $S \in \mathcal{K}$ set $\overline{\mathcal{S}} = \bigcup_{F \in \mathcal{S}} F \subset \mathbb{E}^n$ is said to be the *union of* $\mathcal{S}$.

$\mathcal{K}$ can be described as a collection of simplices that might be linked by $(k-l)$-faces from itself. This also allows for *islands*, that is: simplicial sub-complexes with all simplices linked to at least one other simplex by a $(k-l)$-face.

However, the fully-geometric definition of a simplicial complex (Definition 1.1.6) is too restrictive to cover meshes in practice. A counter-example seen in Fig. 1.2 (b) depicts some intersections $K_1 \cap K_2 \neq \varnothing$ which are not contained in the simplicial complex because they do not form its $(k-l)$-faces and, in general, serve no practical purpose.

For such *simplicial pseudo-complexes*, with only some $K_1 \cap K_2 \neq \varnothing$ forming faces of both K_1 and K_2, for K_1, $K_2 \in \mathcal{K}$, we refer to intersections satisfying this condition as *connectivity.* Likewise, faces K_1 and K_2 are said to be *connected* or *adjacent* if they satisfy this condition. As a particular example, a vertex $\mathbf{x}$ is said to be *connected* to $K_1, ..., K_m$ if and only if it is a 0-face of each one of the $(k-l)$-faces.

Simplicial complexes contain only k-simplices for some $k \in \mathbb{N}$, so for $k = 2$ we are limited to triangles only. In order to define higher-order polygons we extend simplicial complexes by unions of adjacent simplices, forming generalized k-dimensional polygons:

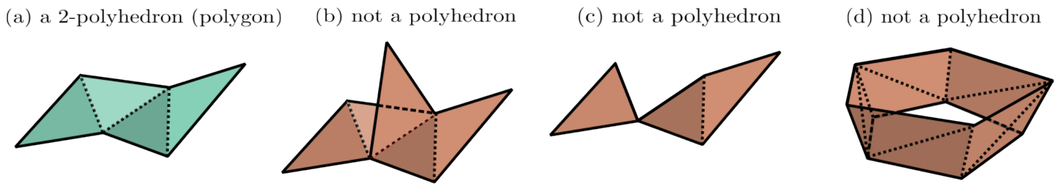


Figure 1.3: Examples of unions of triangles (2-simplices) from which only (a) is a polyhedron with respect to Definition 1.1.7, (b) has a non-manifold edge, (c) contains a non-manifold point, and (d) is homeomorphic to an annulus instead of a disk. *Faux edges* are shown as dotted lines.

Definition 1.1.7. (Polyhedron) Let $K_1, ..., K_r \in \mathcal{K}$ be k-simplices. Union $P = \bigcup_{i=1}^{r} K_i$ is called a *k-polyhedron* if it is homeomorphic to closed k-disc $\overline{\mathbb{D}^k} = \mathbb{D}^k \cup \mathbb{S}^{k-1}$.

Note that by Definition 1.1.7 only some polyhedra are (convex) polytopes. To ensure orientability and reduce the amount of possible configurations, we impose limitations on manifoldness (for counter-examples see Fig. 1.3 (b) and (c)) and restrict ourselves to sets homeomorphic to a closed disk (see Fig. 1.3 (d) for a counter-example). $(k-1)$-faces connecting individual simplices within a polyhedron are referred to as *faux $(k-1)$-faces*[1] because they are used for triangulating polyhedra and are generally not counted among other mesh $(k-1)$-faces.

Definition 1.1.8. (Simplicial and Combinatorial Mesh) A *simplicial* k-dimensional *mesh* is a collection $\mathcal{M}$ of simplices with some $K_1 \cap K_2 \neq \varnothing$ forming faces of both K_1 and K_2, for $K_1, K_2 \in \mathcal{M}$. Moreover, if $\mathcal{M}$ also contains polyhedra P, we say that it is a *polyhedral mesh*. A *combinatorial mesh* is an ordered pair $(\mathcal{M}, V)$ where $\mathcal{M}$ is an abstract simplicial complex (according to Definition 1.1.1) which can also be extended by polyhedral collections of k-simplices.

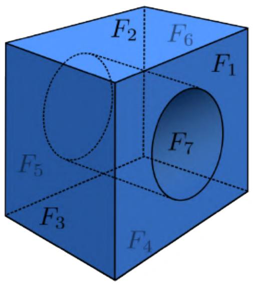


Figure 1.4: An example of a *boundary representation* (BRep) mesh with some faces F_i homeomorphic to an annulus $\mathbb{D}^2 \setminus \mathbb{D}^2_{1>r>0}$.

To declutter notation, we will only use symbol $\mathcal{M}$ instead of the combinatorial ordered pair $(\mathcal{M}, V)$ accompanied by map Φ_V determined by point set $V \subset \mathbb{E}^n$ and also a topological realization. In fact, both the combinatorial and the geometric approach overlap at $\Phi_V(|\mathcal{M}|) = \overline{\mathcal{M}}$ where $\overline{\mathcal{M}}$ is the union of all k-simplices in simplicial $\mathcal{M}$.

Definition 1.1.9. (Triangulation, Tessellation) Let $\mathcal{M}$ be a (simplicial or combinatorial) k-mesh according to Definition 1.1.8. We refer to the subset $\mathcal{M}_k \subset \mathcal{M}$ of all k-simplices such that $\bigcup_{K \in \mathcal{M}_k} K = \overline{\mathcal{M}}$ as a *triangulation of* $\mathcal{M}$, and analogously we refer to a subset $\mathcal{M}_k \subset \mathcal{M}$ of all k-polyhedra satisfying the same condition as a *tessellation*[2] *of* $\mathcal{M}$.

In general, while using the geometric realization $\overline{\mathcal{M}}$ we are not restricted to flat[3] $(k-l)$-faces only. The convex combinations can be replaced with arbitrary $(k-l)$-dimensional interpolations (spline and/or other parametric patches), as it is, for example, in piecewise-smooth surfaces of *boundary representations* (BReps). This extends over to the choice of mappings under which manifolds such as *discs* $\mathbb{D}^{(k-l)}$ are immersed into $\mathbb{E}^n$. We proceed by formally defining such maps extending to closed discs.

Without loss of generality, let $k \leq n$ and $P = \bigcup_{i=1}^{r} K_i$ be a k-dimensional polyhedron composed of k-simplices K_i, then let ∂ be the *boundary operator* mapping points from the interior $\mathrm{Int}(P)$ to boundary

[1]Faux edges are a term used in the VCG Library [38] for polygonal meshes used by MeshLab™.

[2]We chose this name instead of the proper alternative name *polyhedrization* for conciseness. In most literature, *tesselations* are polyhedral partitions of a topological space.

[3]With all principal curvatures equal to zero for every one of their points.

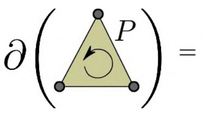


Figure 1.5: The application of boundary operator ∂ on an oriented face P with half edges H_1, H_2, H_3.

∂P. For a $(k-1)$-sphere boundary, ∂ can, for example, map points $\mathbf{x} \neq \mathbf{0}$ from disc $\mathbb{D}^k = \mathrm{Int}(\mathbb{S}^{k-1})$ along a ray from origin $\mathbf{0}$ onto a point intersecting $\mathbb{S}^{k-1}$. The origin $\mathbf{0}$ can then be mapped to any particular point in $\mathbb{S}^{k-1}$. Define $F : X \to \mathbb{E}^n$ with X being a k-manifold with boundary homeomorphic to $\overline{\mathbb{D}^k} = \mathbb{D}^k \cup \mathbb{S}^{k-1}$. This situation can then be described using the following commutative diagram:

$$\begin{tikzcd}
P \arrow[r, "\partial"] & \partial P \\
\mathbb{D}^k \arrow[u, "F"] \arrow[r, "\partial"] & \mathbb{S}^{k-1} \arrow[u, "F"']
\end{tikzcd} .$$

Self intersections of $\mathrm{Im}(F)$ can be avoided if F is an embedding into $\mathbb{E}^n$ instead of an immersion which only guarantees that tangent space $T_{F(x)}P$ is k-dimensional for all $x \in X$.

Definition 1.1.10. (Pseudo-Polyhedral Mesh) A collection $\mathcal{M}$ whose union (in the sense of Definition 1.1.6) $\overline{\mathcal{M}}$ is homeomorphic to the union of a polyhedral combinatorial mesh is said to be a *pseudo-polyhedral mesh.*

A polyhedral simplicial mesh is, of course, also a pseudo-polyhedral mesh.

Boundary representations (BReps), prevalent in CAD geometry (often as a result of constructive solid operations), may also contain faces with interiors not homeomorphic to disks $\mathbb{D}^k$, but rather annuli $\mathbb{D}^k \setminus \mathbb{D}^2_{1>r>0}$ (see Fig. 1.4). The complexity increases if more holes are introduced into a face whose interior becomes homeomorphic to an annulus with m holes: $\mathbb{D}^k \setminus \bigcup_{i=1}^m \mathbb{D}^k_{1>r_i>0,\mathbf{x}_i}, \mathbf{x}_1, ..., \mathbf{x}_m \in \mathbb{D}^k, \overline{\mathbb{D}^k_{1>r_i>0,\mathbf{x}_i}} \subset \mathbb{D}^k, \overline{\mathbb{D}^k_{1>r_i>0,\mathbf{x}_i}} \cap \overline{\mathbb{D}^k_{1>r_j>0,\mathbf{x}_j}} = \varnothing$ for $i \neq j$ and $i, j \in \{1, ..., m\}$.

The complexity of BReps reaches outside the scope of this work, and we can safely assume we are working with polyhedral meshes. Additionally, since non-manifold or non-orientable polyhedral meshes are not used as input for our experiments, we safely disregard degenerate differences between the combinatorial and geometric immersion-based representations.

1.1.2 Manifold Meshes

The polyhedral meshes defined in Section 1.1.1 still cover a very broad range of possible geometric realizations. We are often expected to use meshes which approximate smooth surfaces such as spheres, tori, cylinders, or smooth deformations thereof. This means that there needs to be a distinction between mesh geometries restricted by the properties of a subset of a simplicial complex, and those that approach smooth surfaces with finer resolution.

Definition 1.1.11. (Manifold Mesh) Let X be a topological k-manifold, and $F : X \to \mathbb{E}^n$ an immersion. Let $\mathcal{M}$ be a polyhedral k-mesh. If $F[X] = \overline{\mathcal{M}}$ then $\mathcal{M}$ is said to be a *manifold polyhedral mesh.*

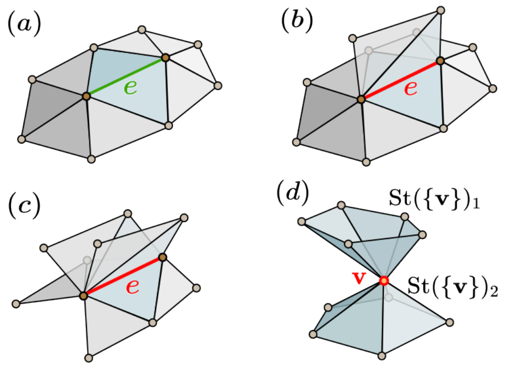


Figure 1.6: (a): A 2-manifold triangle mesh with edge e having exactly 2 adjacent faces. (b): A *T-junction* type non-manifold edge e with 3 adjacent faces. (c): A non-manifold edge e with 4 adjacent faces. (d): A non-manifold vertex $\mathbf{v}$ with two adjacent stars $\mathrm{St}(\{\mathbf{v}\})_i,\ i = 1, 2$.

Recall that a k-manifold X is a topological space which can be locally described as a linear space $\mathbb{R}^k$. If there exist points in the geometric realization $\overline{\mathcal{M}}$ which cannot be uniquely mapped onto $\mathbb{R}^k$ (via charts) the *manifoldness* assumption is violated. Such regions are, of course, some $(k-l)$-faces for $l = 1, ..., k$ in $\mathcal{M}$ which we refer to as *non-manifold* $(k-l)$*-faces.*

Definition 1.1.12. (Non-Manifold Edges and Vertices) Let $\mathcal{M}$ be a polygonal 2-mesh. If edge $e \in \mathcal{M}$ is shared by more than 2 faces $P_e^{(0)}, P_e^{(1)} ..., P_e^{(r)} \in \mathcal{M}$, we refer to it as a *non-manifold edge*. Furthermore, let $\mathbf{v} \in \overline{\mathcal{M}}$ be a vertex in $\mathcal{M}$. If a star $\mathrm{St}(\{\mathbf{v}\})$ of 0-face $\{\mathbf{v}\}$ is a union of more than one distinct stars: $\mathrm{St}(\{\mathbf{v}\})_i, i = 1, ..., r$ each with union $\overline{\mathrm{St}(\{\mathbf{v}\})_i}$ homeomorphic to a closed disc $\overline{\mathbb{D}}^2$, we say that $\mathbf{v}$ is a *non-manifold vertex* in $\mathcal{M}$.

The presence of non-manifold edges or vertices usually foretells issues in use, for example, in 3D-printing. If the underlying printing framework cannot uniquely interpret what subset of ambient space should be an interior of a solid object $\overline{\mathcal{M}}$, the printer will either reject the model, or attempt to print it with significant errors. A wide variety of mesh processing tools therefore offers procedures for healing non-manifold meshes. The process of healing, of course, depends on the type and severity of a non-manifold artifact. Both non-manifold edges and vertices can then be further categorized using orientation.

1.1.3 Orientation and Half-Edges

Much like the choice of an orientation of a coordinate system for describing a physical process, the choice of $[\varpi]$ is fundamental to the way the rest of a mesh-processing application is implemented. The order in which vertices of a mesh are, for example, written to a file is foundational precursor to the subsequent construction of the mesh itself. In the practical context of most implementations, there is preference for positive orientation with further use, for example, in the right-hand rule to compute face normals from boundary vertices. Since we interact with physical objects in the world from their exterior, the agreed upon direction of normals to surfaces is, naturally, *outward-pointing*. Hence converting all faces so that all their normals are outward-pointing is a practical preprocessing step available for the users of a graphics or simulation application.

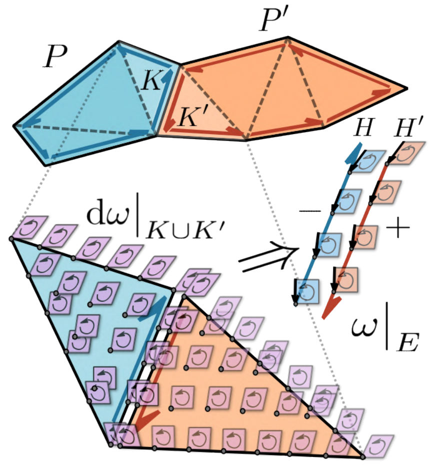


Figure 1.7: A 2-volume form $\mathrm{d}\omega \in [\varpi]$ defined on $K \cup K' \subseteq P \cup P'$ is induced by 1-forms ω on $E = H = H'$ with opposite signs.

Definition 1.1.13. (Orientation) Let P be a k-polyhedron for $k \leq n$, then the equivalence class $[\varpi]$ of volume forms on P is called an *orientation* of P.

Definition 1.1.14. (Mesh Orientation) Let $\mathcal{M}$ be a manifold mesh without boundary in $\mathbb{E}^n$ then its orientation $[\varpi]$ is defined if it is the same for all $P \in \mathcal{M}$. $[\varpi]$ is said to be positive, if normal vectors $\mathbf{n} \in \mathbb{R}^n$ to points $\mathbf{x} \in P \in \mathcal{M}$ are outward-pointing to $\overline{\mathcal{M}} \cup \mathrm{Int}(\overline{\mathcal{M}})$ where $\overline{\mathcal{M}}$ is the union of $\mathcal{M}$. If $\overline{\mathcal{M}}$ is a manifold with boundary, it inherits orientation $[\varpi]$ from $\partial\overline{\mathcal{M}}$.

Orientation of polyhedra is also reflected in $(k-l-1)$-*half-faces* $H \subset P$ of $(k-l)$-polyhedra P forming a chain $\partial P = \bigcup_{H \subset \partial P} H$ of $(k-l-1)$-simplices, each with orientation, of course, matching that of P (see Fig. 1.5).

Theorem 1.1.1. *(Existence of an Opposite Half-Face) Let P be a $(k-l)$-polyhedron for $l = 0, 1, ..., k-2$ of a k-manifold mesh $\mathcal{M}$ with (without loss of generality positive) orientation. If H is a $(k-l-1)$-half-face of oriented $(k-l)$-polyhedron P and P is sharing a $(k-l-1)$-face (edge) E with another $(k-l)$-face P' with the same orientation as P, then there exists exactly one $(k-l-1)$-half-face H' such that $H = H' = E$ but with opposite orientation.*

Proof. Let $K \subseteq P$ be a $(k-l)$-simplex contained in a $(k-l)$-polyhedron P. Since $\overline{\mathcal{M}}$ is a k-manifold, there is exactly one polyhedron P' and one simplex $K' \subseteq P'$, so we have two neighboring simplices K and K'

sharing a $(k-l-1)$-face E. From the definition of orientation in Def. 1.1.14, we choose a $(k-l)$-volume form $\mathrm{d}\omega \in [\varpi]$ defined on both K and K'. Since $\mathrm{d}\omega$ is nowhere-vanishing, orientations $\omega\big|_H$ and $\omega\big|_{H'}$ on $H = H' = E$ induced by $\mathrm{d}\omega$ are well defined with opposite signs[4], and thus cancel out (see Fig. 1.7). Therefore, by coupling $(H, \omega\big|_H)$ and $(H', \omega\big|_{H'})$ we construct a unique pair of opposite $(k-l-1)$-half-faces. □

Definition 1.1.15. (Adjacency Maps) Let $\mathcal{M}$ be a 2-manifold mesh (possibly with boundary). Let h be a 1-half-face, that is: a *half-edge* of 2-face $P \in \mathcal{M}$ then

(1) $\mathrm{Opp} : h \mapsto h'$ maps h to its opposite half-edge in the sense of Theorem 1.1.1.

(2) $\mathrm{Vert} : h \mapsto \mathbf{v}$ where $\mathbf{v}$ is a unique *tail vertex* of h, and $\mathrm{Vert}^{-1} = \mathrm{He} : \mathbf{v} \mapsto h$ its *outgoing half-edge.*

(3) $\mathrm{Next} : h \mapsto h_n$ and $\mathrm{Prev} : h \mapsto h_p$ where h_n, h_p are *next* and *previous half-edges* of $h \subset P$ in P sharing vertices with h so that $\mathrm{Next}(\mathrm{Prev}(h)) = \mathrm{Prev}(\mathrm{Next}(h)) = h$. Furthermore, we define $\mathrm{Next}(\mathbf{v}) = \mathrm{Vert}(\mathrm{Next}(\mathrm{He}(\mathbf{v})))$ and similarly $\mathrm{Prev}(\mathbf{v}) = \mathrm{Vert}(\mathrm{Prev}(\mathrm{He}(\mathbf{v})))$ to obtain the *next* and *previous* vertices in a polygon respectively.

(4) $\mathrm{Face} : h \mapsto P$.

Clearly, map Vert is a bijection between the set $V_P = \{\mathbf{v}_{P_1}, ..., \mathbf{v}_{P_m}\}$ of vertices of polygon P and its half-edges $H_P = \{h_{P_1}, ..., h_{P_m}\}$. Hence, we can *circulate* over vertices $\mathbf{v}_{P_1}, ..., \mathbf{v}_{P_m}$ without half-edges, using just ordered m-tuples of vertices representing a combinatorial polygon P. If we choose a single *starting half-edge* $h_0 \subset P$ (and likewise a *starting vertex* $\mathbf{v}_0$) we can also define $\mathrm{Vert} : P \mapsto h_0$.

The preference for tail vertex is merely a convention which can be inverted to a *head vertex* of each half-edge. For example, the PMP Library [69] stores head vertex of each half-edge in the implementation of `SurfaceMesh` [68]. Tail vertex can then be accessed by map $\mathrm{Vert}_{\mathrm{head}} \circ \mathrm{Opp}$ where $\mathrm{Vert}_{\mathrm{head}}$ maps a half-edge h to its head vertex. PMP also reduces the total number of references by making sure every half-edge is stored right before or after its opposite. The Geometry Central library [65], on the other hand, stores references to tail vertices.

Half-edges with tail and head vertices form a complete loop. Because of that there exists $m \geq 3$ such that $\mathrm{Prev}^m(h) = h$ and $\mathrm{Next}^m(h) = h$. This value is sometimes referred to as the *valence of face P*, analogous to the valence of a vertex counting the number of adjacent edges [69]. The existence of previous and next half-edges, h_p and h_n, for each half-edge h is implied by the boundary orientation $\omega\big|_{\partial P}$ induced from volume form $\mathrm{d}\omega$ on P via an outward-pointing normal vector field to P. Since half-edges form a chain of oriented 1-faces of a polygon, such members can be found even for degenerate (non-manifold) loops with valence 1 (a single self-referencing half-edge) or 2 (a polygon with 2 vertices). Besides conflicting with manifoldness (see Definition 1.1.11), these configurations are usually also discarded because they contain no useful geometric information and prevent the viability of further computations which assume non-degeneracy.

1.1.4 Data Structures for Representing Meshes

Clearly, the definitions formulated in Section 1.1.1 need to be applied in a fully discrete computational setting, which naturally gives rise to the preference for combinatorial definition equipped with vertex coordinates representing the size and shape of the geometric structure. However, simplicial complexes $\mathcal{K}$ and polyhedral meshes $\mathcal{M}$ contain, by definition, all connectivity information between any pair of $(k-l)$-faces for $l = 0, 1, ..., k$. This is clearly too complex for any data structure or a file with reasonable size, thus alternative approaches to store connectivity information are used. Since we focus on 2-dimensional surface meshes, we

[4]More specifically: $(k-l-1)$-forms with opposite sign are induced by applying $\mathrm{d}\omega$ to outward-pointing normal vector fields to simplices K and K'.

only need to store primary information about polygons and vertices with some additional properties such as surface normals and texture coordinates. The inter-polygonal connectivity information is optional, but not used, for example, in standard file formats.

Polygon Soup

The *polygon soup* approach completely omits connectivity information (see Fig.1.8 (a)), and stores additional properties attached to a polygon such as face normal. Although in the mathematical setting the 2-simplices (triangles) might intersect with other faces at edges, they are stored independently from one another with each facet containing the coordinates of its vertices. This is the case for the STL file format [1, 71] which also allows binary encoding to save storage space because of the large memory requirements for meshes with many triangles. Polygon soup data can then be converted to a representation that actually shares polygon boundary points by removing duplicates[5] as in the VCG Library[6] [38] used by MeshLab™.

Buffer Mesh

Buffer-based mesh data requires, on the contrary, much less memory since it contains a *vertex table* containing the coordinates of presumably unique points and a *face vertex index table* which only references the boundary points in the vertex table via indices (see Fig.1.8 (b)). This approach is actually the closest to the one using data $(\mathcal{M}, V)$ of abstract connectivity information $\mathcal{M}$ (indices) and geometric realization as a list of points V. Normal and texture coordinates can also be stored in a table and then referenced by indices in the vertex index table. A notable example of such approach is the commonly used Wavefront OBJ file format [72] with indices starting from 1.

Half-Edge Mesh

Although the above techniques contain connectivity within individual polygons, adjacency between faces and vertices outside the face loop needs to be computed when necessary. A *half-edge representation* of mesh $\mathcal{M}$, described by Campagna et al. [16] at the time as a *directed edge representation* for surface meshes, contains all necessary information for direct access to mesh primitives adjacent to a reference primitive, that is: vertex, edge, or a face. Because it requires more than a minimal amount of information to be stored, unlike the previously mentioned polygon soup and buffer representations, the half-edge representation exists only within an application without a standardized file format. As evident from the name, the fundamental

[5] Also referred to as *welding* close-enough vertices

[6] In function `RemoveDuplicateVertex` implemented in file `vcg/complex/algorithms/clean.h` in the library repository.

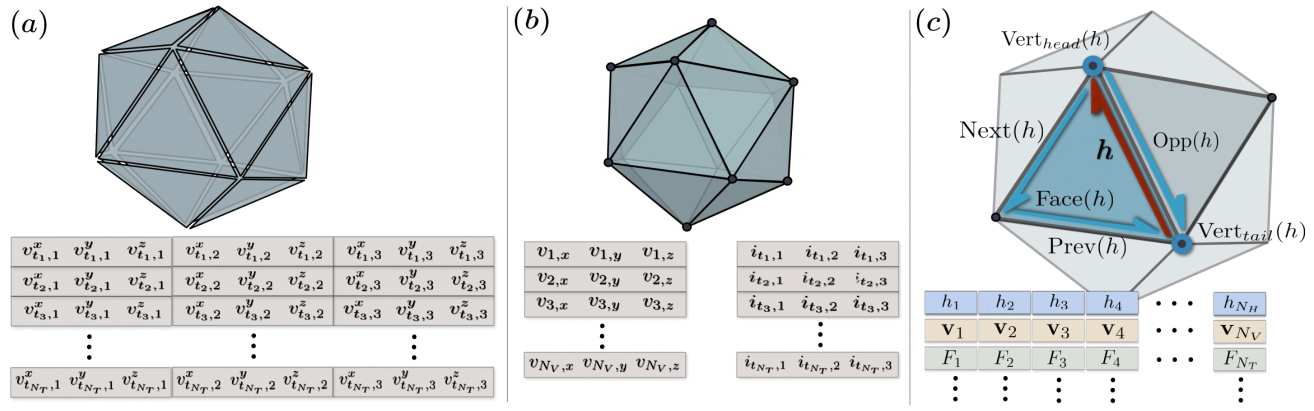

(a)
$v^x_{t_1,1}$ $v^y_{t_1,1}$ $v^z_{t_1,1}$ $v^x_{t_1,2}$ $v^y_{t_1,2}$ $v^z_{t_1,2}$ $v^x_{t_1,3}$ $v^y_{t_1,3}$ $v^z_{t_1,3}$
$v^x_{t_2,1}$ $v^y_{t_2,1}$ $v^z_{t_2,1}$ $v^x_{t_2,2}$ $v^y_{t_2,2}$ $v^z_{t_2,2}$ $v^x_{t_2,3}$ $v^y_{t_2,3}$ $v^z_{t_2,3}$
$v^x_{t_3,1}$ $v^y_{t_3,1}$ $v^z_{t_3,1}$ $v^x_{t_3,2}$ $v^y_{t_3,2}$ $v^z_{t_3,2}$ $v^x_{t_3,3}$ $v^y_{t_3,3}$ $v^z_{t_3,3}$
$v^x_{t_{N_T},1}$ $v^y_{t_{N_T},1}$ $v^z_{t_{N_T},1}$ $v^x_{t_{N_T},2}$ $v^y_{t_{N_T},2}$ $v^z_{t_{N_T},2}$ $v^x_{t_{N_T},3}$ $v^y_{t_{N_T},3}$ $v^z_{t_{N_T},3}$
(b)
$v_{1,x}$ $v_{1,y}$ $v_{1,z}$
$v_{2,x}$ $v_{2,y}$ $v_{2,z}$
$v_{3,x}$ $v_{3,y}$ $v_{3,z}$
$v_{N_V,x}$ $v_{N_V,y}$ $v_{N_V,z}$
$i_{t_1,1}$ $i_{t_1,2}$ $i_{t_1,3}$
$i_{t_2,1}$ $i_{t_2,2}$ $i_{t_2,3}$
$i_{t_3,1}$ $i_{t_3,2}$ $i_{t_3,3}$
$i_{t_{N_T},1}$ $i_{t_{N_T},2}$ $i_{t_{N_T},3}$
(c)
Vert$_{head}(h)$
Next(h)
h
Opp(h)
Face(h)
Vert$_{tail}(h)$
Prev(h)
h_1 h_2 h_3 h_4 h_{N_H}
$\mathbf{v}_1$ $\mathbf{v}_2$ $\mathbf{v}_3$ $\mathbf{v}_4$ $\mathbf{v}_{N_V}$
F_1 F_2 F_3 F_4 F_{N_T}

Figure 1.8: Representations of mesh data of an icosahedron: polygon soup (a), buffer-based (b), and half-edge (c).

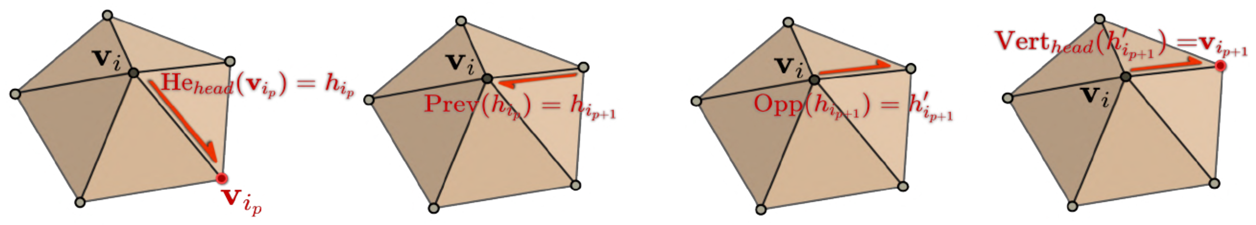

$\mathbf{v}_i$
$\mathrm{He}_{head}(\mathbf{v}_{i_p}) = h_{i_p}$
$\mathbf{v}_{i_p}$
$\mathbf{v}_i$
$\mathrm{Prev}(h_{i_p}) = h_{i_{p+1}}$
$\mathbf{v}_i$
$\mathrm{Opp}(h_{i_{p+1}}) = h'_{i_{p+1}}$
$\mathrm{Vert}_{head}(h'_{i_{p+1}}) = \mathbf{v}_{i_{p+1}}$
$\mathbf{v}_i$

Figure 1.9: A visualization of circulating from vertex $\mathbf{v}_{i_p}$ to the next vertex $\mathbf{v}_{i_{p+1}}$ in the neighborhood $\mathcal{N}(\mathbf{v}_i)$ of a central vertex $\mathbf{v}_i$.

building block of this data structure is a half-edge. In the last few paragraphs of Section 1.1.3, we remark that its implementations might vary between one storing tail or head vertex reference.

Besides vertices and half-edges, polygons can also be stored in contiguous arrays (see Fig.1.8 (c)) each with only a reference to a single (base) half-edge. Libraries such as Geometry Central [65] also store *boundary loops* with the same implementation as polygon faces, but marked as a representation referring to a chain of exterior half-edges encircling a single boundary $\partial\overline{\mathcal{M}}_i,\ i \in \mathbb{N}$ out of possibly many. On the contrary, the PMP Library [69] only marks exterior half-edges with a boundary property. In fact, all data referred to as *properties* is stored in a property template described, for example, by Botsch et al. [12] to improve performance. A similar template-based approach is used in CGAL [74].

Most libraries with a half-edge mesh implementation provide the use of *circulators* - special iterators across the contiguous arrays of half-edges, vertices, faces etc. for iterating along elements adjacent to a particular primitive [69, 74]. A notable example is a *vertex-vertex circulator*, iterating along all vertices connected to a single central vertex with edges. Half-edges and their adjacency information (see Definition 1.1.15) are, of course, the essential building block for such operation. If $\mathbf{v}_i, 1 \leq i \leq N_V$ is a central vertex and $\mathbf{v}_{i_p} \in \mathcal{N}(\mathbf{v}_i),\ 1 \leq p \leq m = |\mathcal{N}(\mathbf{v}_i)|$ is a vertex in its neighborhood $\mathcal{N}(\mathbf{v}_i)$, then we can switch from $\mathbf{v}_{i_p}$ to the next vertex $\mathbf{v}_{i_{p+1}} \in \mathcal{N}(\mathbf{v}_i)$ in the counter-clockwise direction by taking $\mathbf{v}_{i_{p+1}} = \mathrm{Vert}_{head}(\mathrm{Opp}(\mathrm{Prev}(\mathrm{He}_{head}(\mathbf{v}_j))))$ where He_{head} maps $\mathbf{v}_j$ to a half-edge pointing towards it, and Vert_{head} maps a half-edge to its head vertex (see Fig. 1.9). Similarly to regular iterators, circulators require references to a *begin* and an *end* item. For a full cycle, we simply set both references to a single vertex $\mathbf{v}_{i_1}$. Circulation stops at an element before the end vertex reference regardless of whether central vertex $\mathbf{v}_i$ is at a boundary loop or not.

1.1.5 Operations Changing Mesh Tessellation

For reasons, not limited just to decimation or the improvement of polygon quality (for example, see [64]), changes to mesh connectivity and also the overall tessellation of the mesh surface are of substantial importance in many applications. In this section, we formalize the most common operations such as *split*, *subdivision*, *flip*, *collapse*, etc. for general k-meshes. While considering such operations we investigate meshes and their connectivity, making *topological changes* to the combinatorial structure itself.

In practice, the topological changes of $\mathcal{M}$ are usually interpreted in the combinatorial sense (for graph vertex tuples) rather than changes in the *topology* of set $\overline{\mathcal{M}}$. From the continuous point of view, we can choose between uncountably many families $\tau_{\overline{\mathcal{M}}}$ of subsets of $\overline{\mathcal{M}}$ satifsfying the conditions of a topology.

A straight-forward example is a Euclidean topology inherited from the ambient space. We can easily concieve of a subdivision operation on a flat polyhedron which covers the same set of points in space even though the connectivity changes. This means we can have changes to mesh structure which might not affect the point-set topology at all. Moreover, $\mathcal{M}$ might contain naked vertices or edges which renders the purely topological definition invalid (a simplicial k-mesh might not have a topology). For this reason we refer to the notion of *tessellation* from Definition 1.1.8.

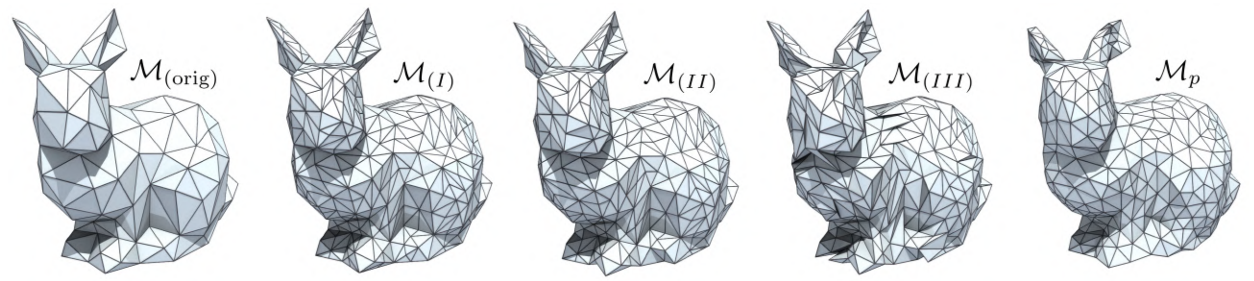


Figure 1.10: One iteration consisting of four steps of *uniform remeshing* [10, 69] applied to input triangle mesh $\mathcal{M}_{(\text{orig})}$ with starting mean edge length $\bar{l} = 14.9$ to target edge length $l = 8.5$: split long edges $\mathcal{M}_{(I)}$, collapse short edges $\mathcal{M}_{(II)}$, flip edges to equalize valence $\mathcal{M}_{(III)}$, and finally we have the post-processed $\mathcal{M}_p$ after 5 steps of tangential relaxation (tangential movement of vertices without combinatorial changes).

Let $\mathcal{M}, \mathcal{M}'$ be polyhedral k-meshes. A map $\mathcal{M} \mapsto \mathcal{M}'$ where $\mathcal{M}'$ may contain simplicial and connectivity information different from $\mathcal{M}$ is called a *tessellation-changing operation on mesh* $\mathcal{M}$. Note that such definition is broad-enough to include maps between very different meshes. Even a map which maps a mesh to an empty mesh or a mesh without triangulation is contained in the set of all tessellation-changing operations. In practice, we need to restrict ourselves to a subset of minimal operations affecting only a small-enough subset of polyhedra. More complex operations will be a result of repeated application of minimal operations.

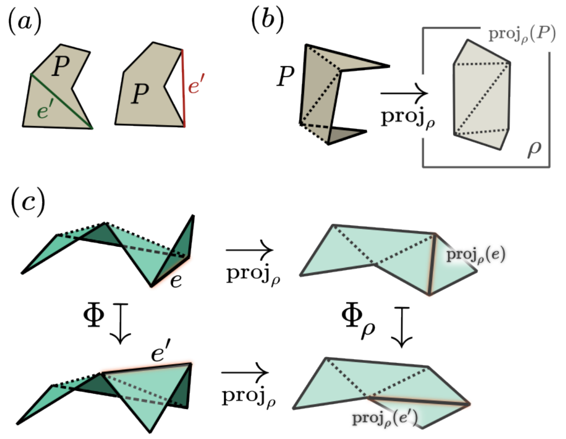


Figure 1.11: (a): Edge e' *diagonal* in P and *non-diagonal* in P. (b): Projection of polygon P into plane ρ. (c): Flip of edge e to e' which is ρ-diagonal in its ambient polygon.

Flip

Edge flips have been a standard procedure for transforming planar triangulations. As the matter of fact, Lawson [46] shows that any planar triangulation can be transformed to another using a finite sequence of flips, in 1972. However, for planar tessellations composed of k triangulated polygons each with n vertices, flipping leads to general complexity of $\mathcal{O}(kn)$ of such transformation [35]. Cheng and Jin [18] show that for a class deemed most representative of meshes encountered in practice, the complexity reduces to $\mathcal{O}(N_V)$ where N_V is the number of vertices. The latest novel use of edge flips can be seen, for example, with Sharp and Crane [64] who develop a framework for finding intrinsic geodesic paths with the help of edge flipping.

Definition 1.1.16. (Diagonal) Let P be a planar polygon. Edge e connecting two of its boundary vertices $\mathbf{v}_1, ..., \mathbf{v}_m$ is said to be *diagonal* in P if $\text{Int}(e) \subset \text{Int}(P)$.

For illustration, see Fig. 1.11 (a).

Definition 1.1.17. (Edge Flip Between Triangles) Let $\mathcal{M}$ be a triangle 2-mesh, and let T, T' be triangles sharing an edge e. A tessellation-changing operation $\Phi : \mathcal{M} \mapsto \mathcal{M}'$ is called a *flip of edge* e if T and T' are replaced with T'' and T''' and edge e by e' connecting vertices from T and T' from outside of e.

If triangles T and T' lie in the same plane we have $T \cup T' = T'' \cup T'''$. In particular, for co-planar pairs of adjacent triangles T and T', edge e flipped to e' is a diagonal in $T \cup T' = T'' \cup T'''$, that is: $\text{Int}(e) \subset \text{Int}(T \cup T')$

and analogously $\mathrm{Int}(e') \subset \mathrm{Int}(T'' \cup T''')$. Without co-planarity of all points in all convex sets T and T', the notion of a diagonal needs to be extended with the help of projections:

Definition 1.1.18. An edge e is said to be ρ-*diagonal* in polygon P if there exists a plane $\rho \subset \mathbb{E}^3$ and a projection $\mathrm{proj}_\rho : \mathbb{E}^3 \to \rho$ such that $\mathrm{Int}(\mathrm{proj}_\rho(e)) \subset \mathrm{Int}(\mathrm{proj}_\rho(P \cup P'))$ and $\mathrm{proj}_\rho(P \cup P')$ is a polygon in ρ.

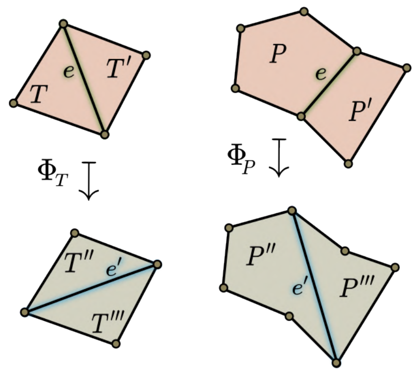


Figure 1.12: Flip Φ_T flips edge e between two adjacent triangles T and T', and Φ_P flips an edge between two adjacent general polygons P and P'.

In other words: the image $\mathrm{proj}_\rho(P \cup P')$ is homeomorphic to a closed disk $\overline{\mathbb{D}}^2$ and from the definition of a projection: so are its constituent triangles $\mathrm{proj}_\rho(T_i), i = 1, ..., r$. In some exotic cases, projecting onto a plane (see Fig. 1.11 (b)) is insufficient, and we require, for example, more general projections (e.g: stereographic etc.). Rest assured, we shall work with well-behaved polygons which can be planarized by projecting onto planes $\rho \subset \mathbb{E}^3$ (see Fig. 1.11 (c)). Furthermore, we can extend the notion of edge flipping to higher-valence polygons P and P' by considering flips between diagonals of a polygon $P \cup P'$.

Definition 1.1.19. (Edge Flip Between Polygons) Let $\mathcal{M}$ be a polygonal 2-mesh, and let $P, P' \in \mathcal{M}$ be polygons adjacent at edge e. A tessellation-changing operation $\Phi : \mathcal{M} \mapsto \mathcal{M}'$ is called a *flip of edge* e if P, P' can be replaced with a pair of polygons P'', P''' sharing an edge e', ρ-diagonal in $P \cup P'$.

Boundary edges $e \in \partial\overline{\mathcal{M}}$ of triangles T are, by definition, not flippable since there is no other diagonal edge in T to which we can transform the configuration. For a boundary polygon P a diagonal e' can be found by shifting the pair of vertices, for example, in clockwise or counter-clockwise direction. However, there will always be at least one vertex $\mathbf{v}_e$ which needs to be discarded from P (see Fig. 1.13 (a)). Furthermore, if e has a neighboring boundary edge $\tilde{e}$ also in P, $\mathbf{v}_e$ can become a naked vertex in $\mathcal{M}$ (see Fig. 1.13 (b)). Besides the combinatorial obstacles, flipping boundary edges fails also on the basis of the resulting flipped polygon, not covering the same points of its constituent triangle convex hulls.

Bern et al. [7] extend the notion of flip transformations to hexahedral meshes used, for example, in finite element analysis. Although in this work, we focus on flips on surface meshes only, for curious readers we can extend the notion to higher dimensions:

Definition 1.1.20. (Edge Flip Between Polyhedra) Let $\mathcal{M}$ be a polyhedral k-mesh, and let $P, P' \in \mathcal{M}$ be polyhedra adjacent to $(k-1)$-face E. A tessellation-changing operation $\Phi : \mathcal{M} \mapsto \mathcal{M}'$ is called a *flip of* $(k-1)$-*face* E if P, P' can be replaced with a pair of polyhedra P'', P''' sharing a $(k-1)$-face E' such that there exists a hyperplane $\Pi \subset \mathbb{E}^n$ and a projection $\mathrm{proj}_\Pi : \mathbb{E}^n \to \Pi$ such that $\mathrm{Int}(\mathrm{proj}_\Pi(e)) \subset \mathrm{Int}(\mathrm{proj}_\Pi(P \cup P'))$ and $\mathrm{proj}_\Pi(P \cup P')$ is a polyhedron in Π.

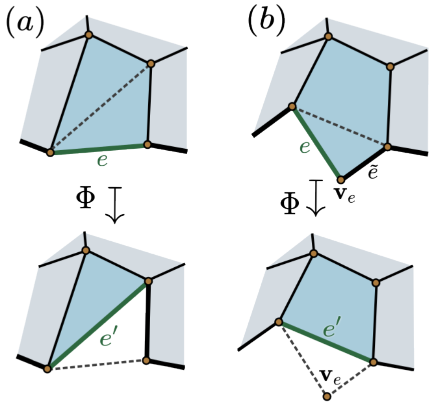


Figure 1.13: (a): Flipping a single boundary edge e. (b) Flipping two consecutive boundary edges e and $\tilde{e}$.

Split and Subdivision

Flipping, described in previous section, does not change the total amount of $(k-l)$-faces, $l = 0, 1, ..., k$ and their connectivity information in k-mesh $\mathcal{M}$. It changes valence of adjacent vertices. However, it is often necessary to perform operations which either reduce the amount of information in $\mathcal{M}$ by removing $(k-l)$-faces, or to add $(k-l)$-faces increasing mesh complexity. This section focuses on some from the latter class of tesselation-changing operations, taking a $(k-l)$-face as input and outputting multiple $(k-l)$-faces.

Definition 1.1.21. (Triangle Edge Split) Let $\mathcal{M}$ be a triangle 2-mesh, and e an edge with two adjacent triangles $K, K' \in \mathcal{M}$. A tessellation-changing operation $\Xi : \mathcal{M} \mapsto \mathcal{M}'$ is called a λ-*split of edge* e if this edge is replaced by edges $e^{(0)}$ and $e^{(1)}$ such that $e = e^{(0)} \cup e^{(1)}, \text{Int}(e^{(0)}) \cap \text{Int}(e^{(1)}) = \varnothing$ and for $\lambda \in (0,1)$ we have $\mu_1(e^{(0)}) = \lambda \mu_1(e)$ and $\mu_1(e^{(1)}) = (1-\lambda)\mu_1(e)$ where μ_1 is the 1-dimensional Lebesgue measure (edge length). Added vertex $\mathbf{v}_e^{\Xi}$ such that $\{\mathbf{v}_e^{\Xi}\} = e^{(0)} \cap e^{(1)}$ is referred to as *split vertex*. For triangles K and K', Ξ satisfies one of two alternatives:

(1) the adjacent triangles K and K' are replaced with quadrilateral polygons P and P' sharing split vertex $\mathbf{v}_e^{\Xi}$ and covering the same points in $\mathbb{E}^3$, that is: $K = P$ and $K' = P'$, or

(2) K and K' are replaced with $K_{(0)}, K_{(1)}$ and $K'_{(0)}, K'_{(1)}$ respectively, such that $K = K_{(0)} \cup K_{(1)}$ and $K' = K'_{(0)} \cup K'_{(1)}$, with *split edges* $\tilde{e} = K_{(0)} \cap K_{(1)}$ and $\tilde{e}' = K'_{(0)} \cap K'_{(1)}$.

The third option without any straight-forward practical use would be to replace K and K' with their union, and split edge e into $e^{(0)}$ and $e^{(1)}$ while discarding all connectivity information. Splitting a boundary edge $e \in \partial\overline{\mathcal{M}}$ reduces the amount of new faces in case (2) of Definition 1.1.21 to a half. Furthermore, parameter $\lambda \in (0,1)$ is generally set to 1/2 in most applications, that is: edge e is split at its midpoint. For example, function `SurfaceMesh::split` from the PMP library [69] with an edge input parameter uses midpoint split.

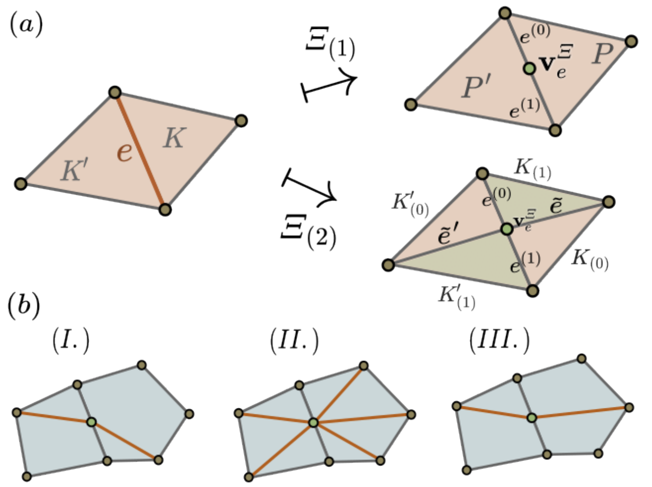


Figure 1.14: (a) Standard edge splits of e: the formation of polygons P and P' from triangles ($\Xi_{(1)}$), and split with splitting faces using split edges $\tilde{e}$ and $\tilde{e}'$ ($\Xi_{(2)}$). (b) Polygon face-splitting edge split types (I.) - (III.).

Definition 1.1.22. (Polygon Edge Split) Let $\mathcal{M}$ be a polygonal 2-mesh, and e an edge with two adjacent polygons $P, P' \in \mathcal{M}$. A tessellation-changing operation $\Xi : \mathcal{M} \mapsto \mathcal{M}'$ is called a λ-*split of edge* e if this edge is replaced by edges $e^{(0)}$ and $e^{(1)}$ analogously to Definition 1.1.21. In addition to alternatives (1) and (2) of Definition 1.1.21 extended to polygons, we define the following alternatives for splitting faces P and P':

(1) the adjacent polygons P and P' are split by chosen edges $\tilde{e}$ and $\tilde{e}'$ respectively,

(2) P and P' are split by edges $\tilde{e}_1, ..., \tilde{e}_{m-l}$, $0 \le l \le (m+1)(m-2)/2$ and $\tilde{e}'_1, ..., \tilde{e}'_{m'-l'}$, $0 \le l' \le (m'+1)(m'-2)/2$, ρ-diagonal in P and P' respectively[7]. m and m' are the valences of P and P' respectively (before adding split vertex $\mathbf{v}_e^{\Xi}$ to each) and l, l' are the numbers of split edges which are not ρ-diagonal in each polygon.

(3) Additionally, we can consider a combination between options (1) and (2) by considering only some edges ρ-diagonal in each polygon.

[7]It can be shown by induction, that the number of diagonal edges in a convex planar m-gon is $m(m-2)/2$. Therefore, by adding an extra vertex, to a general polygon gives rise to an upper bound $(m+1)(m-2)/2$ for the number of diagonals.

Analogously to the extension of edge flip to a $(k-1)$-face (Definition 1.1.20) we can extend the above definitions to splits of $(k-1)$-faces shared by polyhedra P and P'. Edge splits of type (1) with respect to Definition 1.1.21 lead to the notion of a *face split* which may follow a general pattern: given vertices $\mathbf{v}_1, ..., \mathbf{v}_m$ of P, if we can construct an edge e, ρ-diagonal in P connecting any pair of vertices, say $\mathbf{v}_i$ and $\mathbf{v}_j$ for $i \neq j$ and $i, j \in \{1, ..., m\}$, then as long as split edges of the face do not intersect in $\text{Int}(P)$ we can split polygon P into multiple sub-faces. Naturally, we are then not limited by the pre-existing polygon vertices, if we simply add new vertices to ∂P connecting split edge endpoints. We shall first define this notion for general polyhedral meshes:

Definition 1.1.23. (Face Subdivision) Let $\mathcal{M}$ be a polyhedral k-mesh and P its k-face. A tesselation-changing operation $\Sigma : \mathcal{M} \mapsto \mathcal{M}'$ such that P is decomposed into polyhedra $P_1, ..., P_d \in \mathcal{M}'$ is called a $d:1$ *subdivision*[8] *of face* P. Furthermore, let $P_1, ..., P_d \in \mathcal{M}'$ be polyhedra satisfying $\text{Int}(P_i) \cap \text{Int}(P_j) = \varnothing$ for $i \neq j$, then a subdivision Σ satisfies one of three alternatives:

(1) the adjacent k-faces change their valence without adding new edges outside of $\text{Int}(P)$,

(2) we also subdivide each adjacent k-face using ρ-diagonal edges to account for the vertices added to ∂P, or

(3) we discard the connectivity of P to other faces and just subdivide P treating it as an independent polyhedron in a polyhedron soup representation. After subdivision P is replaced by $P_1, ..., P_d$.

If $\mu(P_i) = \mu(P_j), i \neq j$ and $i, j \in \{1, ..., d\}$ where μ is a k-dimensional Lebesgue measure induced by scalar and cross product in $\mathbb{E}^n$, then subdivision Σ producing $P_1, ..., P_d$ is said to be *uniform*. Let P and P' be adjacent polyhedra in $\mathcal{M}$. Subdivision Σ_P of P is *compatible with* $\Sigma_{P'}$ of P' if adjacent polyhedron P' requires no additional subdivision of type 2. A *global subdivision* Σ of mesh $\mathcal{M}$ is a collection of subdivisions $\Sigma_{P_1}, ..., \Sigma_{P_{N_F}}$ applied to each face $P_i \in \mathcal{M}, i = 1, ..., N_F$. We say that Σ is *globally compatible* if it is global, and compatible for each pair of connected polyhedra P and P'.

[8]Read d-to-1 subdivision.

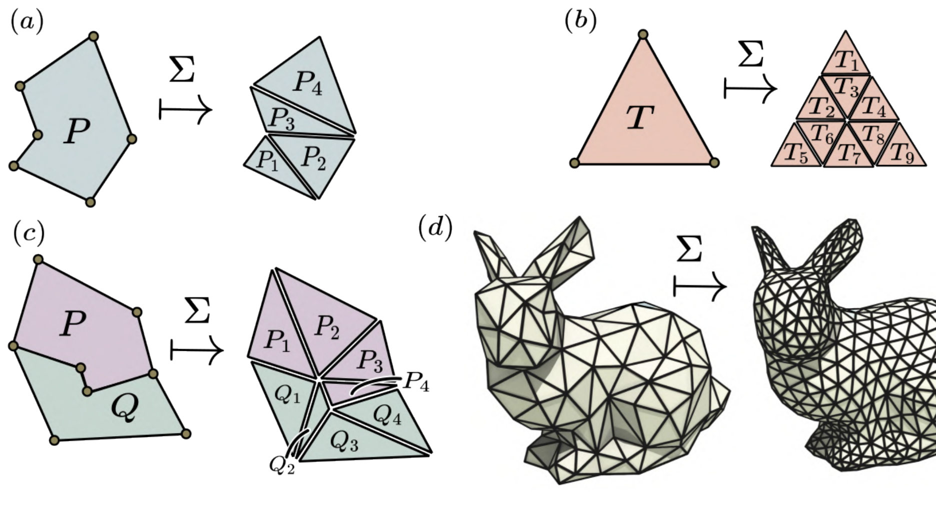


Figure 1.15: (a): A $4:1$ subdivision of polygon P. (b): A uniform $9:1$ subdivision of a triangle T. (c): Subdivision Σ composed of two compatible $4:1$ subdivisions on polygons P and Q. (d): A globally-compatible approximating uniform $4:1$ subdivision (Loop [50]).

Notice that we do not require that $\bigcup_{i=1}^{d} P_i = P$ because the union of decomposed polyhedra might occupy different points in $\mathbb{E}^n$. The equivalence of $\bigcup_{i=1}^{d} P_i$ to P takes the form of homeomorphism in $\mathbb{E}^n$. This means that not only the newly added vertices in $\mathcal{M}'$ may occupy positions different from the points in original polygon, but also the positions of original vertices in ∂P can be updated. The formulas describing the update of pre-existing and the creation of new vertices in a mesh, accompanied by an update of connectivity, under subdivision Σ will be referred to as a *subdivision scheme*.

Definition 1.1.24. (Approximating and Interpolating Subdivision) Let $\mathcal{M}$ be a polyhedral k-mesh and $\Sigma : \mathcal{M} \mapsto \mathcal{M}'$ a $d:1$ subdivision on $\mathcal{M}$. Let $V_P = \{\mathbf{v}_1, ..., \mathbf{v}_m\} \subset \partial P$ be the set of boundary vertices of the subdivided polyhedron $P \in \mathcal{M}$. If $\Sigma\big|_{V_P} = \mathrm{id}\big|_{V_P}$ then we refer to Σ as an *interpolating* subdivision. On the other hand, if Σ changes positions of vertices in V_P, we call it an *approximating* subdivision (see Fig. 1.16).

A well-defined scheme of subdivision Σ allows for repeated subdivision of original polygons. An attentive reader might then ask: What happens to surface $\overline{\mathcal{M}}$ when repeatedly applying a globally compatible $d:1$ subdivision ad infinitum?

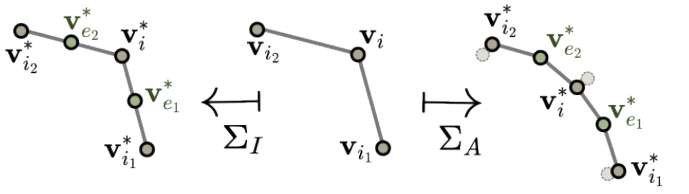


Figure 1.16: A 2D slice showing an interpolating subdivision Σ_I and an approximating subdivision Σ_A.

Definition 1.1.25. (Limit Surface) Let $\mathcal{M}$ be a 2-mesh, and Σ a globally compatible $d:1$ subdivision. Set

$$\overline{\left(\lim_{s\to\infty} \Sigma^s(\mathcal{M})\right)} \subset \mathbb{E}^3 \ , \quad \text{where } \Sigma^s = \overbrace{\Sigma \circ ... \circ \Sigma}^{s\text{-times}}$$

is called the *limit surface* of subdivision Σ.

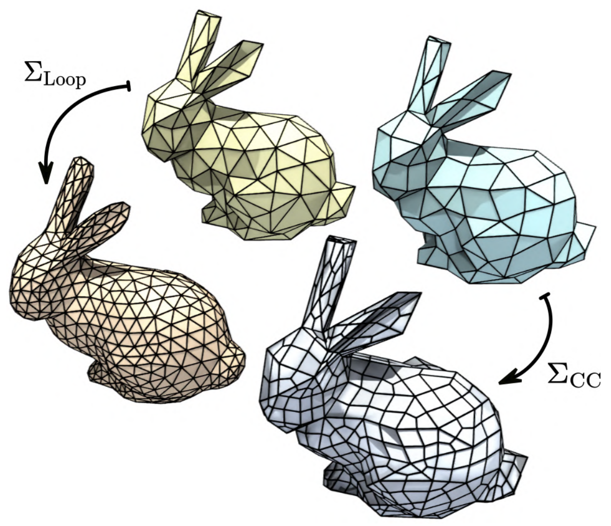


Figure 1.17: The Loop subdivision scheme Σ_{Loop} and the Catmull-Clark variant Σ_{CC}.

Commonly used subdivision schemes are globally compatible. Even if one chooses to subdivide a subset $\overline{\mathcal{M}}_\Sigma \subset \overline{\mathcal{M}}$ of mesh $\mathcal{M}$ compatibility at faces adjacent to $\partial\overline{\mathcal{M}}_\Sigma$ should be enforced to make sure the result mesh $\mathcal{M}'$ keeps its key geometric characteristics such as genus or Euler characteristic $\chi(\overline{\mathcal{M}})$.

A notable example of a globally compatible 4:1 approximating subdivision of triangle 2-meshes by Loop [50] which generates C^2 continuous limit surfaces everywhere except at extraordinary vertices[9] where they are C^1 continuous (see Fig. 1.15 (d), and Fig. 1.17). In particular, upon subdivision, we distinguish between old vertices updated to:

$$\mathbf{v}^* = (1-\beta)\mathbf{v} + \beta \sum_{p=1}^{m} \mathbf{v}_{i_p}, \qquad (1.1)$$

with

$$\beta = \frac{5}{8} - \left(\frac{3}{8} + \frac{1}{4}\cos\left(\frac{2\pi}{m}\right)\right)^2$$

[9]If $\mathcal{M}$ has vertex valence m for almost all pairwise neighboring vertices, an *extraordinary vertex* is one with valence $m' \neq m$.

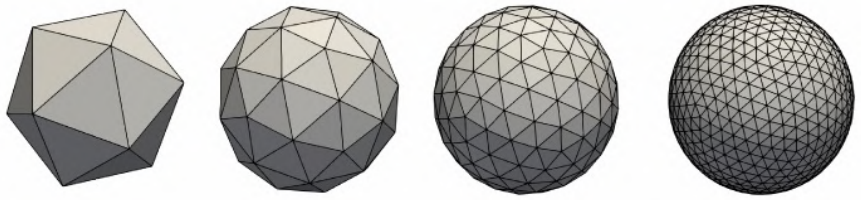


Figure 1.18: A globally-compatible interpolating 4:1 subdivision scheme constructing icosahedral tessellations of a sphere.

where $\mathbf{v}_{i_p} \in \mathcal{N}(\mathbf{v}), p \in \{1, ..., m = |\mathcal{N}(\mathbf{v})|\}$, and for each edge e new vertices are computed by:

$$\mathbf{v}_e^* = \frac{1}{4}\bigg(3(\mathbf{v} + \mathbf{v}_{i_p}) + \mathbf{v}_{i_{p+1}} + \mathbf{v}_{i_{p-1}}\bigg), \text{ where } \mathbf{v}, \mathbf{v}_{i_p} \in e \tag{1.2}$$

and $\text{conv}(\{\mathbf{v}, \mathbf{v}_{i_{p-1}}, \mathbf{v}_{i_p}\})$ and $\text{conv}(\{\mathbf{v}, \mathbf{v}_{i_p}, \mathbf{v}_{i_{p+1}}\})$ are two triangles from vertex $\mathbf{v}$ and its neighbors $\mathcal{N}(\mathbf{v})$ sharing edge e. Boundary edge vertices are simply computed as average of their endpoints: $\mathbf{v}_e^* = (\mathbf{v} + \mathbf{v}_{i_p})/2$, whereas boundary old vertices are updated using:

$$\mathbf{v}^* = \frac{1}{4}\big(6\mathbf{v} + \mathbf{v}_{i_p} + \mathbf{v}_{i_{p+1}}\big).$$

An interpolating[10] analogue which holds old vertices constant, and inserts midpoints $\mathbf{v}_e^* = (\mathbf{v} + \mathbf{v}_{i_p})/2,\ \mathbf{v}, \mathbf{v}_{i_p} \in \mathbb{S}^2,\ p \in \{1, ..., 5\}$ to each edge e can be used to construct an *icosahedral tessellation of a sphere*. We start with an icosahedron, and insert vertices $\mathbf{v}_e^*$ for each edge e which we then project onto $\mathbb{S}^2$ by normalizing:

$$\text{proj}_{\mathbb{S}^2}(\mathbf{v}_e^*) = \mathbf{v}_e^* / \|\mathbf{v}_e^*\|.$$

We can then scale and translate the resulting mesh to match an approximation of an arbitrary sphere $\mathbb{S}^2_{\mathbf{c},r} \subset \mathbb{E}^3$ with center $\mathbf{c} \in \mathbb{E}^3$ and radius $r > 0$ (see Fig. 1.18). We will refer to both Loop subdivision and the construction of an icosahedral sphere from the combinatorial point of view in Section 1.1.7.

Another example of an approximating scheme is the Catmull-Clark globally compatible subdivision [17] sampling new vertices at face centroids applicable to arbitrary polygons. The Catmull-Clark approach does not subdivide all polygons into the same number of faces, instead the amount of subdivision faces depends on valence m_P of an individual face $P \in \mathcal{M}$ (see Fig. 1.17). We can therefore refer to it as a $m_P : 1$ subdivision.

Although there are modern attempts to extend the known subdivision schemes to non-manifold meshes (see Moulaeifard et al. [55]), clearly not all meshes $\mathcal{M}$ contain only $(d : 1)$-subdivisible faces. Naked edges and vertices need to be cleared before we can apply a practical globally compatible subdivision scheme.

Subdivision surfaces - a class of 2-meshes generated via a subdivision scheme (according to Definitions 1.1.23 and 1.1.24) from simple input meshes - is widely used to generate finer approximations of smooth surfaces. Such meshes are then used, for example, in the final high *level of detail* (LOD) renderings of character shapes in animation [22]. Due to high demands on performance, however, animated characters in interactive real-time applications usually use meshes with fixed polygon tessellation.

Edge Collapse and Vertex Split

In this section, we focus on two mutually inverse tessellation changing operations on 2-meshes. They represent an atomic approach to add or remove a vertex with a chosen edge as a basis.

[10]Under *spherical interpolation*.

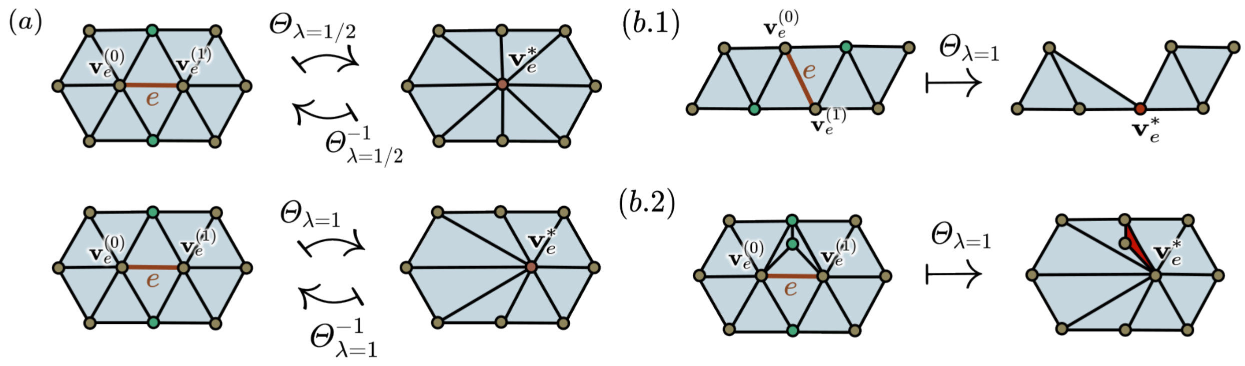


Figure 1.19: (a): Collapse Θ_λ of edge e and its inverse operation Θ_λ^{-1}, the vertex split, parametrized by $\lambda \in [0, 1]$. (b.1), (b.2): Cases when shared valence does not hold the values in (1.3) producing a non-manifold vertex (b.1), and a non-manifold edge to a triangle with flipped orientation (b.2).

Definition 1.1.26. (A Collapsible Edge) Let $\mathcal{M}$ be an oriented 2-manifold polygonal mesh, possibly with a boundary $\partial\overline{\mathcal{M}}$. An edge $e \in \mathcal{M}$ with endpoints $\mathbf{v}_e^{(0)}$ and $\mathbf{v}_e^{(1)}$ is *collapsible* if:

$$|\mathcal{N}(\mathbf{v}_e^{(0)}) \cap \mathcal{N}(\mathbf{v}_e^{(1)})| = \begin{cases} 1\,, & \text{for } e \subset \partial\overline{\mathcal{M}} \\ 2\,, & \text{otherwise} \end{cases} \tag{1.3}$$

and for a polygon P adjacent to e only one of vertices $\mathrm{Prev}(\mathbf{v}_e^{(0)}), \mathrm{Next}(\mathbf{v}_e^{(1)}) \notin e$ in P can be a boundary vertex.

Value in formula (1.3) will be referred to as *shared valence* of edge endpoints $\mathbf{v}_e^{(0)}$ and $\mathbf{v}_e^{(1)}$.

Definition 1.1.27. (Edge Collapse) Let $\mathcal{M}$ be an oriented 2-manifold polygonal mesh, possibly with a boundary $\partial\overline{\mathcal{M}}$. A tessellation-changing operation $\Theta : \mathcal{M} \mapsto \mathcal{M}'$ is referred to as *collapse of edge* $e \in \mathcal{M}$ if edge e, collapsible with respect to Definition 1.1.26, is removed from $\mathcal{M}$ with its endpoints $\mathbf{v}_e^{(0)}$ and $\mathbf{v}_e^{(1)}$ identified with a single vertex $\mathbf{v}_e^* \in e$. Adjacent polygons $P^{(0)}$ and $P^{(1)}$ are updated correspondingly.

To ensure compatibility with polygons $P^{(0)}$ and $P^{(1)}$ adjacent to e half-edges h_0 and h_1 are removed from their corresponding half-edge loops. If the size of the resulting half-edge loop for any of the neighboring polygons $P^{(0)}$ and $P^{(1)}$ drops below 2, the corresponding face is removed. If $e \in \partial\overline{\mathcal{M}}$ then only one adjacent polygon P is affected.

Without restrictions formulated in Definition 1.1.26, changes induced by edge collapse Θ might lead to the construction of non-manifold vertices or edges (see Fig. 1.19 (b.1) and (b.2) respectively). In particular, the shared valence restriction prevents the formation of a non-manifold edge connecting the resulting vertex $\mathbf{v}_e^*$ with an extra shared neighborhood vertex $\tilde{\mathbf{v}} \in \mathcal{N}(\mathbf{v}_e^{(0)}) \cap \mathcal{N}(\mathbf{v}_e^{(1)})$. The latter restriction on vertices $\mathrm{Prev}(\mathbf{v}_e^{(0)}), \mathrm{Next}(\mathbf{v}_e^{(1)}) \not\subset e$, on the other hand, prevents the formation of non-manifold vertices formed upon removal of edge e.

As in Definition 1.1.21, the position of vertex $\mathbf{v}_e^*$ can be parametrized by a linear interpolation parameter $\lambda \in [1, 0]$. In general, collapse Θ of edge e is the *midpoint collapse* for $\lambda = 1/2$. According to [69], for $\lambda = 0$, we refer to Θ as a *collapse of half-edge* $h^{(0)} = \mathrm{He}_{head}(\mathbf{v}_e^{(0)})$, and analogously for $\lambda = 1$, Θ collapses half-edge $h^{(1)} = \mathrm{He}_{head}(\mathbf{v}_e^{(1)})$.

Definition 1.1.28. (Vertex Split) Let $\mathcal{M}$ be an oriented 2-manifold polygonal mesh possibly with a boundary $\partial\overline{\mathcal{M}}$. A vertex $\mathbf{v} \in \mathcal{M}$ is said to be *splittable* if there exists an edge collapse $\Theta : \mathcal{M}' \mapsto \mathcal{M}$ such that edge $e \in \mathcal{M}'$ with endpoints $\mathbf{v}_e^{(0)}$ and $\mathbf{v}_e^{(1)}$ is collapsed to vertex $\mathbf{v}$. A tesselation-changing operation $\Theta^{-1} : \mathcal{M} \mapsto \mathcal{M}'$ is then referred to as *vertex split*.

One can also consider cases when vertex split Θ^{-1} is combinatorially equivalent to the split Ξ of edge $e \in \mathcal{M}$. The equivalence holds only when the newly added vertex $\mathbf{v}^* \in e$ for Ξ ends up with valence corresponding to the valence of an endpoint $\mathbf{v}_{\tilde{e}}^{(j)}, j \in \{0,1\}$ of edge $\tilde{e}$ collapsed by Θ.

Among the more "direct" operations which add or remove mesh vertices is, for example, *vertex removal*, that is: remove a vertex $\mathbf{v}$ from $\mathcal{M}$ and all simplices and polyhedra in its star $\mathrm{St}(\{\mathbf{v}\})$. An inverse operation inserting a vertex would need a well-defined desired neighborhood $\mathcal{N}(\mathbf{v}^*) \subset \overline{\mathcal{M}}$ of the newly inserted vertex while also discarding faces in $\mathrm{Int}(\mathrm{Lk}(\{\mathbf{v}^*\}))$ to avoid creating non-manifold edges (where Lk is the link according to Definition 1.1.1).

Since operations Θ (from Definition 1.1.27) and its inverse (Definition 1.1.28) can be considered *atomic* (simple enough), they are generally favored over the above-mentioned direct deletion and insertion in various *decimation algorithms*. Moreover, the deletion of vertex $\mathbf{v}$ from $\mathcal{M}$ creates a *hole* from the missing faces in its star $\mathrm{St}(\{\mathbf{v}\})$ which then needs to be properly filled to maintain the topological properties of $\overline{\mathcal{M}}$.

1.1.6 Boundary Loops and Binary Partitioning of the Ambient Space

The missing patches of a surface present problems when the union of mesh $\mathcal{M}$ is treated as a solid object in which we require that there is a clear distinction between its interior $\mathrm{Int}(\overline{\mathcal{M}})$ and exterior $\mathrm{Ext}(\overline{\mathcal{M}})$. This is particularly important in the process of *voxelization* - conversion from mesh representation $\mathcal{M}$ to values in a *voxel grid* $G \subset \mathbb{E}^3$ (for more details see Section 1.2.2). Even with an oriented manifold mesh $\mathcal{M}$, the distinction between the aforementioned subsets of $\mathbb{E}^3$ might be difficult to approximate for large-enough holes.

Definition 1.1.29. (Boundary Edge) Let $\mathcal{M}$ be a manifold 2-mesh with a boundary $\partial\overline{\mathcal{M}}$. Edge $e \subset \partial\overline{\mathcal{M}}$ is referred to as a *boundary* (or *naked*) *edge* if it has only one adjacent face $P \in \mathcal{M}$. A *chain* $e_1, ..., e_m$ of boundary edges such that $e_i \cap e_{i+1} = \{\mathbf{v}_i\},\ i = 1, ..., m$ are *boundary vertices*, is referred to as a *hole*. A mesh $\mathcal{M}$ without holes is said to be *watertight*.

For dimensionality reasons, the holes of $\overline{\mathcal{M}}$ (according to Definition 1.1.29) should not be confused with the holes of the *solid object* $\mathrm{Int}(\overline{\mathcal{M}})$ whose number projects to the *genus* g of $\mathcal{M}$. In particular, for *Euler characteristic* $\chi(\mathcal{M}) = N_V - N_E + N_F$ where N_V is the number of mesh vertices, N_E is the number of edges, and N_F the number of faces in $\mathcal{M}$, we have

$$\chi(\mathcal{M}) = \begin{cases} 2 - 2g\ , & \text{if } \mathcal{M} \text{ is watertight,} \\ 2 - 2g - N_b\ , & \text{otherwise,} \end{cases} \tag{1.4}$$

where N_b is the number of *boundary loops*.

Recall, that in the context of half-edges (see Section 1.1.3) a hole can be represented as a *boundary loop* of chained exterior half edges $h_1, ..., h_m$, treating a hole as a special type of face. True faces adjacent to boundary edges are often referred to as *boundary faces*, and, as we observed in Section 1.1.5, assume fundamentally different behavior for tessellation-changing operations.

Most voxelization techniques require watertight meshes as input. The approach using *angle-weighted pseudonormals* [14] only limits the input to 2-manifold meshes $\mathcal{M}$ for which angle-weighted vertex (pseudo) normals $\hat{\mathbf{n}}_i$ to the mesh surface $\overline{\mathcal{M}}$ can be correctly evaluated at each mesh vertex $\mathbf{v}_i$. Using the *closest point query* for triangles[11] we then interpolate normal vector $\hat{\mathbf{n}}$ at the closest point $\mathbf{v}$ inside the closest triangle using barycentric coordinates. The ambient space can then be partitioned using the sign of the dot product $\hat{\mathbf{n}} \cdot (\mathbf{x} - \mathbf{v})$ where $\mathbf{x} \in \mathbb{E}^3$ is the sampled point (see Fig. 1.20 (b.1), (b.2), and (b.3)).

[11] For example, using `pmp::TriangleKdTree::nearest` from the PMP Library [69].

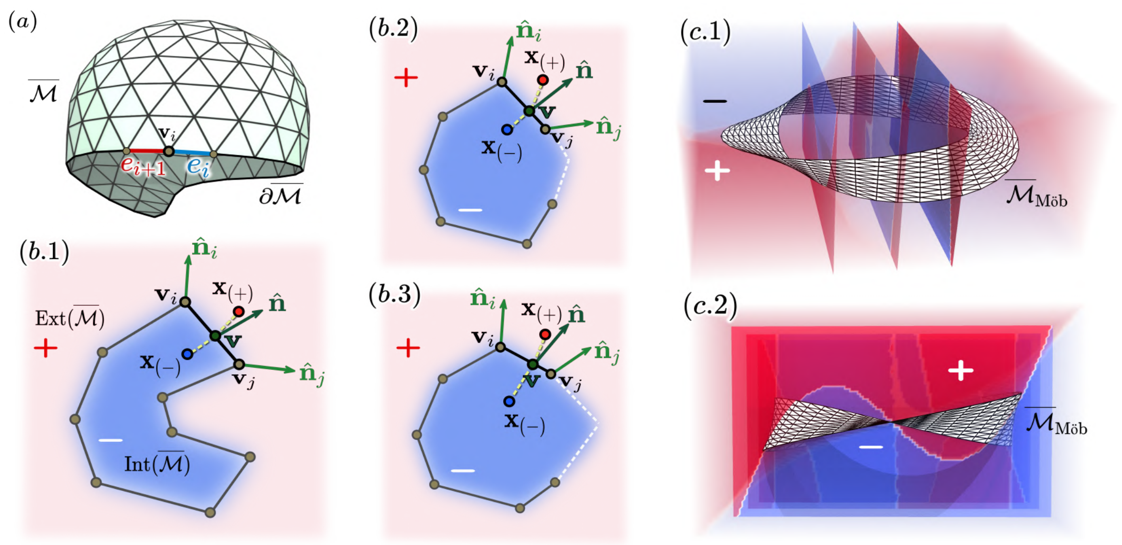


Figure 1.20: (a): Triangle mesh $\overline{\mathcal{M}}$ with a single boundary loop $\partial\overline{\mathcal{M}}$. (b.1): A 2D slice of a watertight mesh $\overline{\mathcal{M}}$ with the sign of dot product $\hat{\mathbf{n}} \cdot (\mathbf{x} - \mathbf{v})$ for any $\mathbf{x} \in \mathbb{E}^3$ partitions the ambient space into interior Int($\overline{\mathcal{M}}$), and exterior Ext($\overline{\mathcal{M}}$). Normal vector $\hat{\mathbf{n}}$ is interpolated from closest face vertex normals $\hat{\mathbf{n}}_i$ and $\hat{\mathbf{n}}_j$. (b.2), (b.3): Introducing a small-enough hole in the mesh does not necessarily undermine the partitioning using the dot product sign. Instead, a cone-like protrusion of negative sign values extends into space (white dotted line). The oblique (c.1) and the side view (c.2) of the sign field for the triangulation of the Möbius strip $\overline{\mathcal{M}}_{\text{Möb}}$ with three slices along the x-axis showing severe discontinuities produced from the incoherent normal vectors to the non-orientable surface.

The partitioning holds even for boundary vertices $\mathbf{v}_i \in \partial\overline{\mathcal{M}}$, the accuracy of the spatial partitioning is, however, limited by the size and shape of the hole. Large boundary loops with highly divergent pseudonormals at their vertices create a cone-like protrusion which extends beyond the non-watertight surface's shape (see Fig. 1.20 (b.2) and (b.3)). The situation reaches extremity for non-orientable surface meshes such as the Möbius strip where the shape resulting from binary partitioning of $\mathbb{E}^3$ becomes highly distorted since it only depends on the closest point query and the relative orientation of the surface pseudonormal (see Fig. 1.20 (c.1) and (c.2)). A solid object representing Int($\overline{\mathcal{M}}$) generated from such partitioning could then easily be non-compact.

It is often desired to work with *solid objects* which represent bulks of material in the real world. For this matter, the methods of *constructive solid geometry* (CSG) require some preprocessing steps to make sure one works with well-defined solids. Whether it is by actually filling the holes of 2-manifold meshes [48], or by introducing variable winding numbers [79], the requirements for the preprocessing technique depend, as always, on the type of input.

1.1.7 The Amount of Triangle Mesh Primitives After $s \geq 0$ Subdivision Steps

Knowing how many vertices, edges or faces a mesh will have after $s \in \mathbb{N}$ successive steps of globally-compatible subdivision (see Definition 1.1.23) becomes crucial, primarily for reasons related to numerical stability of semi-implicit fairing schemes for surface meshes (see Section 2.3), and secondarily to alleviate computational demands for repeated memory allocation. The former rationale, in particular, stems from the fact that under uniform sampling of subdivided surface, one can predict the size of a neighborhood $V \subset \overline{\mathcal{M}}$ containing a single a mesh vertex $\mathbf{v}$. Furthermore, given a family of meshes $\{\mathcal{M}_s\}_{s\in\mathbb{N}_0^+}$ with level of detail (LOD) parametrized by s, one can have, for instance, a more direct access while transitioning between different LODs of a mesh.

Theorem 1.1.2. *Let $\mathcal{M}_s,\ s \in \mathbb{N}_0^+$ be a watertight 2-manifold triangle mesh, and let $\Sigma : \mathcal{M}_{s-1} \mapsto \mathcal{M}_s,\ s > 0$ be a globally-compatible $4:1$ subdivision inserting a single vertex for each edge $e \in \mathcal{M}_s$. Let N_V^s, N_E^s, and N_F^s denote the number of vertices, edges, and faces of $\mathcal{M}_s$ respectively. Then given starting counts N_V^0, N_E^0, and N_F^0 we have:*

$$N_E^s = 4^s N_E^0,\ N_F^s = 4^s N_F^0, \tag{1.5}$$

$$N_V^s = \frac{1}{3}\big(N_E^0(4^s - 1) + 3N_V^0\big). \tag{1.6}$$

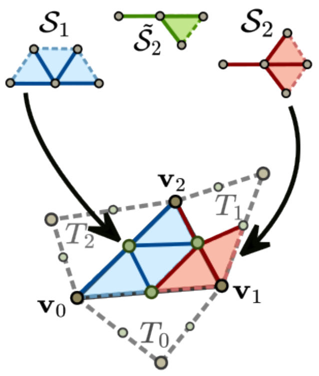


Figure 1.21: Tiling a watertight triangle mesh during 4:1 subdivision.

Proof. First, we consider that Σ subdivides each face into 4 faces, that is $N_F^s = 4N_F^{s-1}$ which yields $N_F^s = 4^s N_F^0$ for any $s \in \mathbb{N}$. However, since we insert a new vertex for each existing edge, the number of added vertices in step s will be equal to edge count N_E^{s-1}. This gives rise to a system of recurrence equations:

$$\begin{aligned} N_V^s &= N_V^{s-1} + N_E^{s-1}, \\ N_E^s &= 4N_E^{s-1}. \end{aligned} \tag{1.7}$$

Before solving this system, we show why the second equation for N_E holds. We need to verify that under Σ the number of edges in $\mathcal{M}_{s-1}$ increases to 4 times the count in previous step.

Since there are no boundary edges in $\mathcal{M}_{s-1},\ s \in \mathbb{N}$, each triangle is guaranteed to have 3 edge-adjacent neighbors T_0, T_1, and T_2. Define $\mathcal{S}_1$ as a quadruple of edges after subdivision stemming from an inserted vertex, and $\mathcal{S}_2$ also a quadruple of created edges distinct by branching from an inserted vertex into a neighboring triangle as well (see Fig. 1.21). We observe, that as long as there is no boundary edge, any triangle can be covered with configurations $\mathcal{S}_1$ and $\mathcal{S}_2$. This pattern can be extended across $\overline{\mathcal{M}_s}$ without conflicts because the valence of any vertex $\mathbf{v}_j,\ j = 0, 1, 2$ is at least 3.

After solving (1.7) using the s-th power of the matrix of the system, we get $N_E^s = 4^s N_E^0$ and (1.6). □

Introducing $N_E^b > 0$ boundary edges implies that the same number of $\mathcal{S}_2$-units needs to be changed to 3-edge configurations $\tilde{\mathcal{S}}_2$ (see Fig. 1.21) reaching only into the interior of one triangle from the previous step $\mathcal{M}_{s-1}$. Due to its inherent dependence on boundary edge count N_E^b, the general recurrence (1.7) is violated.

Similar results were evaluated by Alarcao et al. [3] and Osthoff et al. [56] in preparation of mesh domains in the form of an icosahedral tessellation of a sphere (see Fig. 1.18). The latter approach, however, only holds for the next step of a globally-compatible uniform $N^2 : 1$ subdivision of an icosahedron mesh. The PMP implementation [69] of Loop subdivision[12] pre-allocates mesh memory using recurrence expressed as system:

$$\begin{aligned} N_V^s &= N_V^{s-1} + N_E^{s-1}, \\ N_E^s &= 2N_E^{s-1} + 3N_F^{s-1}, \\ N_F^s &= 4N_F^{s-1}, \end{aligned}$$

with a slightly more complicated solution:

$$\begin{aligned} N_V^s &= \frac{N_F^0}{2}\big(2 - 3 \times 2^s + 4^s\big) + N_E^0(2^s - 1) + N_V^0, \\ N_E^s &= 2^{s-1}\big(3N_F^0(2^s - 1) + 2N_E^0\big), \\ N_F^s &= 4^s N_F^0. \end{aligned}$$

[12]See function `Subdivision::loop` implemented in file `src/pmp/algorithms/Subdivision.h|.cpp` in the library repository.

We can then derive, for example, (1.6) from the first equation using substitution $N_F^0 = \frac{2}{3}N_E^0$ which holds for all watertight manifold triangle meshes. More specifically, every face has 3 half-edges and, by Theorem 1.1.1, there are always two half-edges for each edge, that is: $N_H = 2N_E = 3N_F$. Note that the statement of Theorem 1.1.2 holds for any genus $g \geq 0$, since the introduction of handles in a closed manifold surface $\overline{\mathcal{M}} \subset \mathbb{E}^3$ violates neither the tessellation using units $\mathcal{S}_1$ and $\mathcal{S}_2$ from Fig. 1.21, nor the substitution $N_F^s = \frac{2}{3}N_E^s,\ s \in \mathbb{N}_0^+$.

1.1.8 Quality Metrics

Since meshes might be further used for additional computation and subsequent efficient storage, it is important to measure the *quality* of individual triangles or polygons. The general rule of thumb is that triangle quality is a measure of closeness to the equilateral shape, and analogously the quadrilateral faces should approach flat square shapes.

Faces with large discrepancies in internal angles might produce numerical instability in finite element or finite volume solvers. Yet only a small minority of shapes can be tessellated by regular polygons, that is: we almost always introduce slightly skewed faces. For this reason, we need to evaluate the faces with at least some global metric, and notify the user in case of large deviations from some norm.

An example of a deep investigation for what quality metrics to use can be found in J. Schewchuk's preprint [66]. In this thesis, we will use a small number of quality merics from Knupp [43] which are preferred for example by Sandia National Laboratories in the Cubit Project [62].

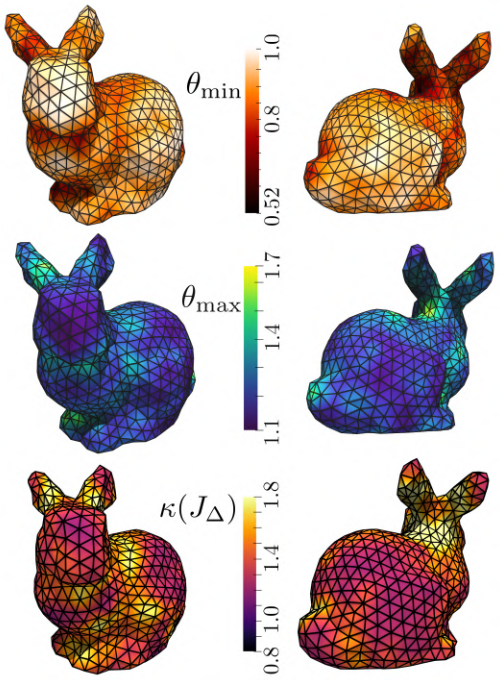


Figure 1.22: Shrink-wrapped (see Chapter 2) *Stanford Bunny* mesh with metrics from Table 1.1 evaluated per triangle and averaged over vertex face stars[13].

The first two metrics in Table 1.1 are self-evident, stemming from the shape of each triangle. The last metric does so as well, with equilateral triangles having $\kappa(J_\Delta) = 1$ and upon skewing of the triangle shape $\kappa(J_\Delta)$ increases to arbitrarily high values.

Since we are working with 2-meshes, the triangles $T = \mathrm{Conv}(\{\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2\})$ embedded in $\mathbb{E}^3$ must be projected into a plane $\mathcal{P}_T$. Define $\overline{\mathbf{v}}_j = \mathrm{proj}_{\mathcal{P}_T}\mathbf{v}_j, j \in \{0, 1, 2\}$. In general, we define a Jacobian by Knupp [43]:

$$J_{\triangleright} = \begin{pmatrix}\overline{\mathbf{v}}_1 - \overline{\mathbf{v}}_0\ , & \overline{\mathbf{v}}_2 - \overline{\mathbf{v}}_0\end{pmatrix} \tag{1.8}$$

and evaluate its *condition number* as:

$$\kappa(J_{\triangleright}) = \|J_{\triangleright}\|\|J_{\triangleright}^{-1}\|, \tag{1.9}$$

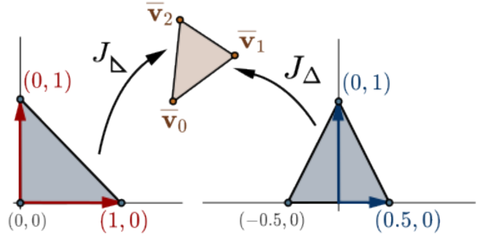


Figure 1.23: Jacobians $J_{\triangleright}$ and J_Δ from unit triangle (left), and the unit equilateral triangle (right) to the planar representation of $T = \mathrm{Conv}(\{\overline{\mathbf{v}}_0, \overline{\mathbf{v}}_1, \overline{\mathbf{v}}_2\})$ in triangle plane: $\mathcal{P}_T$.

where $\|\cdot\|$ is the 1-norm. The Jacobian $J_{\triangleright}$ corresponds to a transformation from a unit triangle $\mathrm{Conv}(\{(0,0)^\top,\ (1,0)^\top, (0,1)^\top\})$ to $T_{\mathcal{P}_T} = \mathrm{Conv}(\{\overline{\mathbf{v}}_0, \overline{\mathbf{v}}_1, \overline{\mathbf{v}}_2\})$ (see Fig.

[13]The *VTK Polydata* format we used supports only values per mesh vertex. Therefore we need to average face values for each vertex to show color plots as in Fig. 1.22.

1.23). We can obtain (1.8) directly from vertices $\mathbf{v}_j \in \mathbb{E}^3,\ j \in \{0,1,2\}$ using edge basis vectors

$$\mathbf{e}_0 = \mathbf{v}_1 - \mathbf{v}_0\ ,\quad \mathbf{e}_1 = \mathbf{v}_2 - \mathbf{v}_0\ ,\quad \mathbf{e}_2 = \mathbf{e}_0 \times \mathbf{e}_1 .$$

to evaluate $J_{◺}$ using orthonormal basis vectors $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$, and $\hat{\mathbf{z}}$, computed as follows: Firstly, $\hat{\mathbf{x}} = \mathbf{e}_0/\|\mathbf{e}_0\|$, and given $\hat{\mathbf{z}} = \mathbf{e}_2/\|\mathbf{e}_2\|$ we get the remaining basis vector by rotating $\hat{\mathbf{x}}$ counter-clockwise about $\hat{\mathbf{z}}$-axis: $\hat{\mathbf{y}} = \hat{\mathbf{x}}^{\perp}_{\hat{\mathbf{z}}}$. This yields:

$$J_{◺} = \begin{pmatrix} \mathbf{e}_0 \cdot \hat{\mathbf{x}} & \mathbf{e}_1 \cdot \hat{\mathbf{x}} \\ \mathbf{e}_0 \cdot \hat{\mathbf{y}} & \mathbf{e}_1 \cdot \hat{\mathbf{y}} \end{pmatrix} = \begin{pmatrix} \|\mathbf{e}_0\| & \mathbf{e}_1 \cdot \hat{\mathbf{x}} \\ 0 & \mathbf{e}_1 \cdot \hat{\mathbf{y}} \end{pmatrix} . \tag{1.10}$$

Base triangle of $J_{◺}$, however, does not correspond to triangles produced by *uniform* or *adaptive remeshing* schemes [10, 23].

To obtain the triangle Jacobian $J_{\triangle}$ modified for equilateral triangles (see Fig. 1.23) we need to transform the corner Jacobian (1.10) to a new basis by scaling it:

$$J_{\triangle} = J_{◺} \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 2\|\mathbf{e}_0\| & \mathbf{e}_1 \cdot \hat{\mathbf{x}} \\ 0 & \mathbf{e}_1 \cdot \hat{\mathbf{y}} \end{pmatrix} ,$$

since the basis of an equilateral triangle is scaled by half in the $\hat{\mathbf{x}}$ direction. Condition number $\kappa(J_{\triangle})$ is then evaluated using 1-norms analogously to (1.9).

Table 1.1: Triangle quality metrics used by [62] with $\kappa(J_{\triangle})$ modified for equilateral triangles.

Metric	Full Name	Full Range	Acceptable Range
$\theta_{\min}$	Min. internal angle	$(0, \frac{\pi}{3})$	$(\frac{\pi}{6}, \frac{\pi}{3})$
$\theta_{\max}$	Max. internal angle	$(\frac{\pi}{3}, \pi)$	$(\frac{\pi}{3}, \frac{\pi}{2})$
$\kappa(J_{\triangle})$	Condition number of triangle Jacobian	$[1, \infty)$	$[1, 1.3)$

We notice, that although they are scale-invariant, the values of $\kappa(J_{\triangle})$ are sensitive to even slight deviations from equilateral shape. Triangles closer to the corner shape ◺ are evaluated with $\kappa(J_{\triangle}) \approx 2$. The upper bound for acceptable values in the Cubit project [62] seems therefore way too strict for example in finite volume schemes of non-linear parabolic models used in Section 2.2.

The notion of extremal angles $\theta_{\min}$ and $\theta_{\max}$ can then be extended to quadrilateral faces with corresponding acceptable ranges. We can then use the corner Jacobian $J_{◺}$ to evaluate skewness of quads, since we require quad faces to be close to square shape.

1.2 Scalar and Vector Fields

Extending our scope to the volumetric subsets of Euclidean space $G \subseteq \mathbb{E}^n$ as domains for geometric data is our next step. In order to avoid the utilization of meshes and boundary representations (BReps), it is necessary to represent the essence of a solid object as values associated with data points in space or alternatively, in a completely implicit form. This can take the form of a function $f : G \to \mathbb{R}$, a vector $\mathbf{v} : G \to \mathbb{R}^n$, or a tensor field. It is important to note that set G is typically a compact simply-connected domain where, for instance, a contour plot of an implicit scalar field can be visualized or where values of a discrete field can be stored.

1.2.1 Functional Representation

Functional representations are a powerful technique for representing shapes in *Constructive Solid Geometry* (CSG). When representing relatively simple (non-fractal) shapes, they can easily be regarded as the least memory-demanding representation. Unlike traditional boundary representations that store explicit geometry, functional representations describe shapes as the composition of a set of functions. These functions

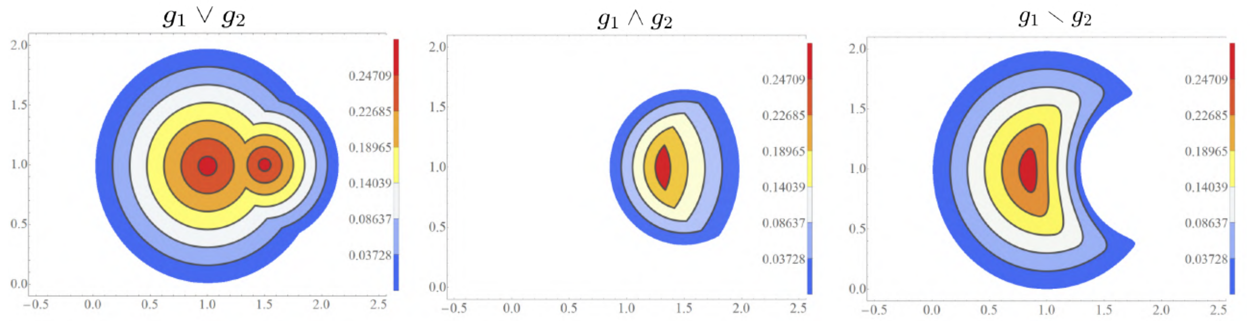


Figure 1.24: Basic boolean operations (1.12) on functions of type (1.11): $g_1 = g(\|\mathbf{x} - \mathbf{c}_1\|/R_1)$ and $g_2 = g(\|\mathbf{x} - \mathbf{c}_2\|/R_2)$ with $R_1 = 1.5$, $R_2 = 1$, $\mathbf{c}_1 = (1,1)^\top$, and $\mathbf{c}_2 = (1.5,1)^\top$.

$f : G \to \mathbb{R}$ define the properties of the shape, such as its (*iso-*) *surface* $\{\mathbf{x} \in \mathbb{E}^3 : f(\mathbf{x}) = 0\}$, *exterior* $\{\mathbf{x} \in \mathbb{E}^3 : f(\mathbf{x}) > 0\}$ and *interior* $\{\mathbf{x} \in \mathbb{E}^3 : f(\mathbf{x}) < 0\}$. The signature of the functional representation can, of course, be opposite. Multiple functions f can be combined using Boolean operations to create complex shapes.

The concept of functional representations was first introduced by Bloomenthal [9], presenting a technique for approximating implicit surfaces using polygonal meshes, which could be efficiently rendered on a computer. The technique relied on a functional representation of the implicit surface, which was defined as the zero set of a set of functions.

A simple functional example representing a primitive shape is the square d^2 of the *Euclidean distance function* from point $(p_x, p_y, p_z)^\top = \mathbf{p} \in \mathbb{E}^3$:

$$d^2_{\mathbf{p}} : \mathbf{x} \mapsto \|\mathbf{x} - \mathbf{p}\|^2 = (x - p_x)^2 + (y - p_y)^2 + (z - p_z)^2 \ , \quad (x, y, z)^\top = \mathbf{x} \in G.$$

Considering only the distance $d_{\mathbf{p}}$, we obtain a simple radially symmetric linear function with a C^1 discontinuity at $\mathbf{p}$. A set with non-zero 2-dimensional Lebesgue measure can then be produced by using $\sqrt{d^2_{\mathbf{p}}(\mathbf{x}) - R^2}$ with radius $R > 0$ of the resulting spherical surface. Besides spheres, one can generate distances to various other geometric primitives whose exact and approximate evaluations can be found, for example, on the website of Inigo Quilez [58].

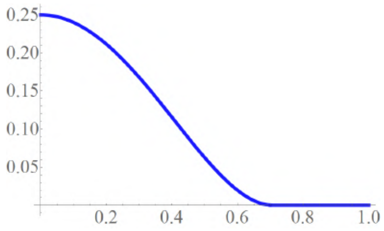


Figure 1.25: A plot of $g(r)$.

Furthermore, Pasko et al. [57] describe the basic set of boolean operations in terms of increasing functions in Section 2.2.1. They even include a variety of smoothing operations as well as blending for the functional values. For our basic example, we take inspiration from Section III. of Ryan Geiss' web article [25] which focuses on the implementation of a feature commonly known as *metaballs* integrated into a variety of graphics engines, that is: for $r > 0$

$$g(r) = \begin{cases} r^4 - r^2 + 0.25 \ , & \text{if } r < 0.7, \\ 0 \ , & \text{otherwise.} \end{cases} \tag{1.11}$$

A metaball with radius $R > 0$ and center $\mathbf{c} \in \mathbb{E}^2$ is represented by function $g(\|\mathbf{x} - \mathbf{c}\|/R)$. The basic boolean operations in this case are:

$$\begin{array}{ll} \text{Union:} & g_1 \vee g_2 = \max(g_1, g_2), \\ \text{Intersection:} & g_1 \wedge g_2 = \min(g_1, g_2), \\ \text{Difference:} & g_1 \setminus g_2 = \max(g_1 - g_2, 0). \end{array} \tag{1.12}$$

The signature of such functions is, of course, such that points $\mathbf{x} \in \mathbb{E}^n$ where $f(\mathbf{x}) > 0$ are in the object's interior. An example of possible results of operations (1.12) between two functions of type (1.11) can be seen in Fig. 1.24.

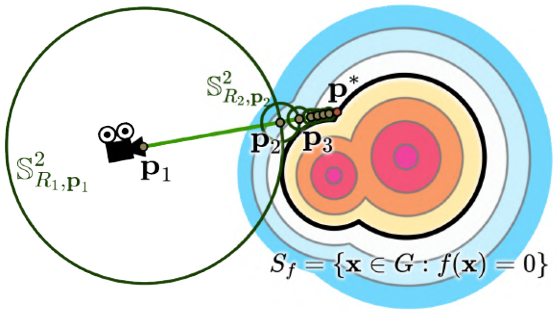

$\mathbb{S}^2_{R_2,\mathbf{p}_2}$
$\mathbf{p}^*$
$\mathbf{p}_2$
$\mathbf{p}_3$
$\mathbb{S}^2_{R_1,\mathbf{p}_1}$
$\mathbf{p}_1$
$S_f = \{\mathbf{x} \in G : f(\mathbf{x}) = 0\}$

Figure 1.26: Ray marching towards an isosurface S_f from point $\mathbf{p}_1$, iterating towards the final point $\mathbf{p}^*$.

The advantage of expressing solid objects as isosurfaces of an implicit scalar field is exploited by the class of rendering algorithms referred to as *ray marching* (or *sphere tracing*) [26]. Instead of computing surface properties from approximating linear elements (triangles), the implicit surface $S_f = \{\mathbf{x} \in G : f(\mathbf{x}) = 0\}$ is rendered as is from an intersection $\mathbf{S}^2_* \cap S_f$ with the final iteration $\mathbf{S}^2_*$ of a series of spheres $\mathbf{S}^2_{\mathbf{p}_i,R_i} \cap S_f \neq \varnothing$, $i = 1, ..., N_{\text{iter}}$ along a ray $\overrightarrow{\mathbf{p}_1\mathbf{p}^*}$. Each iteration of a sphere centered at a point $\mathbf{p}_i$ is computed from the intersection between it an the isosurface S_f. The next point $\{\mathbf{p}_{i+1}\} = \mathbf{S}^2_{\mathbf{p}_i,R_i} \cap \overrightarrow{\mathbf{p}_1\mathbf{p}^*}$ then becomes the centre of the next sphere, until an approximate final intersection point $\mathbf{p}^*$ is reached. The surface properties (normal, colors etc.) are then computed from the intersection with the last sphere $\mathbb{S}^2_*$ with radius below some tolerance limit (see Fig. 1.26).

Functional representations (FReps) are an area of active development. Tereshin et al. [73], for example, extend this notion to *hybrid functional representations* (HFReps) which allow obtaining a continuous smooth distance field in $\mathbb{E}^n$, preserving the advantages of conventional representations presented in this section. The implicit volumetric representations even found use in standardized file formats used in 3D printing, namely the `.3mf` specification with a volumetric extension described in [2].

1.2.2 Voxel Representation

When dealing with volumetric data, one common approach is to represent it in the form of a discrete field defined on a regular grid. This means that the space containing the data is divided into a set of evenly spaced cubes, each of which represents a single unit of data. These cubes, also known as voxels $C_G \subseteq G$, can be thought of as the 3D equivalent of pixels in a 2D image. By organizing the data in this way, it becomes possible to manipulate it using many of the same techniques used in 2D image processing, such as convolution and filtering.

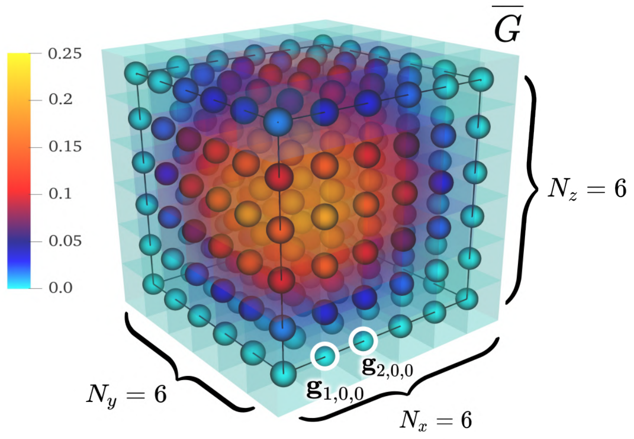


Figure 1.27: Values of function $g_1 \vee g_2$ on regular grid $\overline{G} \subset \mathbb{E}^3$ of cell size $c_G = 1$ with $G = [2.5, 7.5] \times [3.5, 8.5] \times [7.5, 6.5]$ composed of metaballs with radii $R_1 = 4$ and $R_2 = 5$, and centers $\mathbf{c}_1 = (3, 4, 4)^\top$ and $\mathbf{c}_2 = (4, 5, 4)^\top$.

On the other hand, this approach has some drawbacks. One of the main challenges is that the grid-based representation can be quite heavy on memory usage, particularly for large datasets. This is because each voxel requires a significant amount of memory to store its value, and when dealing with volumetric data, there can be millions or even billions of voxels. As a result, techniques like data compression and downsampling are often used to reduce memory requirements while still maintaining a reasonable level of fidelity. Despite these challenges, the standard representation for volumetric data remains the grid-based approach due to its flexibility and the wide range of tools and algorithms available for working with this format.

For the foundational domain of the voxel representation, we define the *regular grid*

$$\overline{G} = \Big\{ \mathbf{g}_{i,j,k} \in G \subset \mathbb{E}^3 \ : \ \mathbf{g}_{0,0,0} = \mathbf{o}_G, \|\mathbf{g}_{i,j,k} - \mathbf{g}\| = c_G, \ \mathbf{g} \in \{\mathbf{g}_{i\pm1,j,k}, \mathbf{g}_{i,j\pm1,k}, \mathbf{g}_{i,j,k\pm1}\} \cap G,$$
$$\text{such that } \mathbf{g}_{i\pm1,j,k} = \mathbf{g}_{i,j,k} \pm (c_G, 0, 0)^\top \text{ and vice versa, for } (i,j,k)^\top \in \mathbb{Z}^{N_x - 1} \times \mathbb{Z}^{N_y - 1} \times \mathbb{Z}^{N_z - 1} \Big\} \quad (1.13)$$

of points with values $f(\mathbf{g}_{i,j,k}) = f_{i,j,k}$. Value $c_G > 0$ will be referred to as *cell size.*

Sampling any scalar field on $\overline{G}$ requires us to store $N_G = N_x \times N_y \times N_z$ floating point values with the *grid dimensions* N_x, N_y, and N_z with floating point cell size $c_G > 0$. To properly define $\overline{G}$ in Euclidean space, we also need to specify the *point of origin* $\mathbf{o}_G \in \mathbb{E}^3$. The *axis-aligned bounding box* $B_G = (\mathbf{b}_{G,\min}, \mathbf{b}_{G,\max})$ (with $\mathbf{o}_G = \mathbf{b}_{G,\min}$) of grid $\overline{G}$ then overlaps with G. If we consider *voxels* volumetrically, as axis-aligned cubes of size c_G, the true extent of the effective volume then reaches further outward into space into each positive and negative direction by the half-size $c_G/2$.

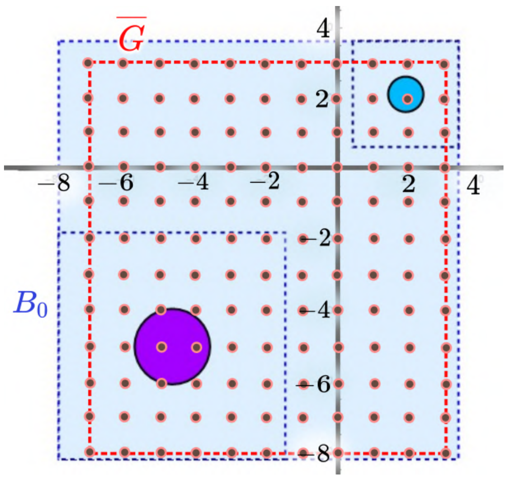


Figure 1.28: An illustration of the global grid $\overline{G}$ constructed from input box B_0 with cell size $c_G = 1$ containing two metaballs with different effective radii.

Because in our applications where mesh surfaces are converted to voxel fields (see Section 1.3.2) we depend heavily on proper alignment of voxels with the imprinted shape. For this reason, we introduce the notion of a *global grid*, namely computing the number of cells as:

$$N_x^- = \lfloor b_{\min,x}/c_G \rfloor, \ N_x^+ = \lceil b_{\max,x}/c_G \rceil, \ N_x = N^+ - N^-$$

for the x coordinate, and likewise for y and z. Operators $\lfloor \cdot \rfloor$, $\lceil \cdot \rceil$ are *floor* and *ceiling* respectively. Then the true bounds B_G of grid $\overline{G}$ have min and max points

$$\mathbf{b}_{G,\min} = c_G(N_x^-, N_y^-, N_z^-)^\top, \mathbf{b}_{G,\max} = c_G(N_x^+, N_y^+, N_z^+)^\top.$$

The resulting grid $\overline{G}$ is therefore guaranteed to be composed of points whose coordinates are integer multiples of the cell size c_G (see Fig. 1.28).

1.3 Conversion Between Geometric Representations

Given all three main types of geometry mentioned in this chapter (see Fig. 1.29), the applications used further in Chapter 2 and potentially in future research, we provide a short description of different types of conversion algorithms. Considering the structure of our current practical application, with regards to the voxelization of mesh inputs, we focus on the computation of *signed distance fields* in more detail (see Section 1.3.2).

1.3.1 Scalar Field Polygonization

We begin with the usually less complex set of conversion methods from both the functional and voxel representations to a polygonal mesh $\mathcal{M}$ (see Definition 1.1.8). Unsurprisingly, the process of meshing a voxel field on a uniform grid $\overline{G}$ is less complicated due to the spatial restraints and finite resolution of the input discrete structure.

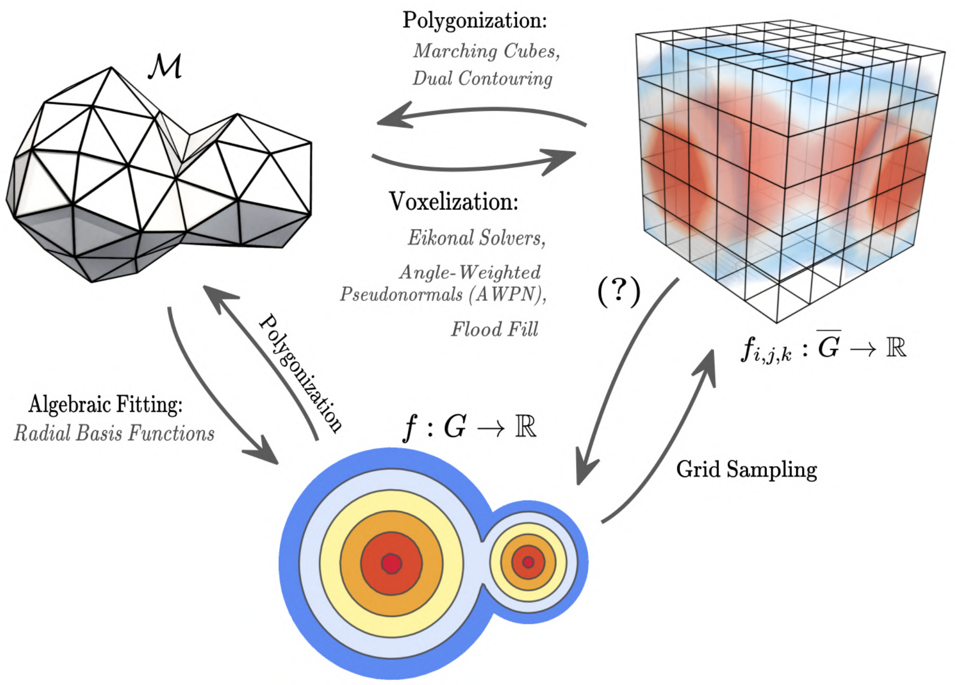


Figure 1.29: Conversion between three main types of geometric representations analyzed in this chapter: polygonal mesh $\mathcal{M}$, scalar voxel field $f_{i,j,k}$, and a functional representation f.

Without regular sampling of the ambient space, the polygonization requires an approach similar to the *advancing front* [67]. The resulting 2-mesh can be a Delaunay triangulation of a manifold surface. In 1996 Hilton et al. [28] were the first to achieve such reconstruction calling it *marching triangles*, while only a few years later Bernardini et al. [8] published a similar result for point cloud inputs, referring to it as the *ball-pivoting* algorithm. Since most practical volumetric data is voxel-based rather than functional, we can move on to a series of polygonization techniques for regular grids $\overline{G}$.

In 1987, Lorensen and Cline published the first algorithm for polygonizing voxel fields [51]. Their *marching cubes* algorithm became perhaps the most widely used surface reconstruction technique of 3D image data, with essential applications in medical imaging. The marching cubes is the most fundamental approach of approximating the *isosurface* $S_f = \{\mathbf{x} \in G : f(\mathbf{x}) = c,\ c \in \mathrm{Im}(f)\}$ of field f, performing a simple linear interpolation on the 8 edges of a single voxel cube $C_G \subset G$ to obtain mesh vertices $\mathbf{v} \in \partial C_G$ on a given edge. Value c will be referred to as *isolevel*.

These vertex configurations are then triangulated according to the 20 fundamental triangulations. More precisely, there are $2^8 = 256$ ways a surface S_f can intersect voxel C_G. Lorensen and Cline [51] have reduced the total number of configurations to 20 using symmetries. For performance reasons, however, almost all implementations of the marching cubes algorithm in modern applications use two specialized *lookup tables*[14] introduced by Paul Bourke in his 1994 blog post[15] [13].

The drawbacks of the marching cubes algorithm lie mainly in both the polygon and the approximation quality of surface S_f, as well as the potential to form non-manifold edges and vertices (see Section 1.1.2). The most notable example of modern techniques handling this issue is the *dual contouring* algorithm, introduced by Ju et al. [40], and further improved to avoid the construction of non-manifold tessellations by Schaefer et al. [63], later made watertight by Rashid et al. [59]. The dual contouring is distinct from the marching cubes

[14]One for edge intersections $S_f \cap \partial C_G$ and one for the vertex indices of the triangulation within C_G.
[15]His `C++` code snippets are sometimes referred to as ”the most copy-pasted code of all time”.

by approximating the normals of the isosurface, and thus allowing sharp features like creases and edges of a solid to be captured in the resulting surface shape (see Fig. 1.30).

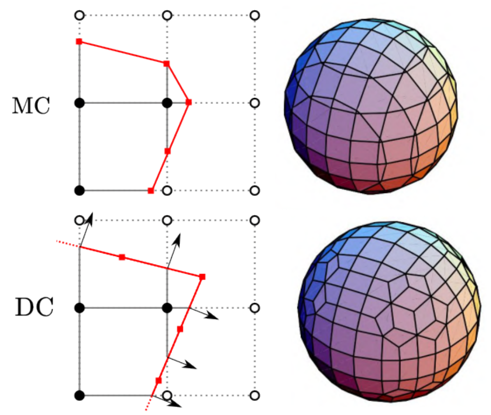


Figure 1.30: The difference between the marching cubes (MC) and the dual contouring algorithm (DC) shown as polygonal voxel slices (source: Ju et al. [40]).

More precisely, the original dual contouring approach is inspired by the *extendend marching cubes* technique looking for sharp feature (dual) vertices $\mathbf{x}$ for each voxel C_G minimizing *quadric error function*

$$E[\mathbf{x}] = \sum_{e=1}^{8} (\hat{\mathbf{n}}_e \cdot (\mathbf{x} - \mathbf{p}_e)) = (\mathbf{Ax} - \mathbf{b})^\top (\mathbf{Ax} - \mathbf{b}),$$

where $\mathbf{A}$ is a symmetric 3x3 matrix and $\mathbf{b}$ a column 3-vector. The quadric error is then minimized using the QR or singular value decomposition [63, 59].

Since we shall use our own feature-detection techniques (see Section 2.4) during exterior surface extraction via *shrink-wrapping*, and also exploit the *adaptive remeshing* method [23] to improve triangle quality (see Section 2.5), the marching cubes polygonization technique will suffice. The remeshing methods themselves will not only provide a robust way to control edge sizing, but also balance out valences of individual vertices.

1.3.2 Mesh Voxelization

The first step in the pipeline of our current application (described in Chapter 2) is the conversion of imported polygonal meshes into voxel fields. Due to the nature of many applications such as 3D printing and collision detection[16], the shape of surface mesh data needs to be propagated into its ambient space. In this section, we focus on some approaches for generating *signed distance fields* from input meshes.

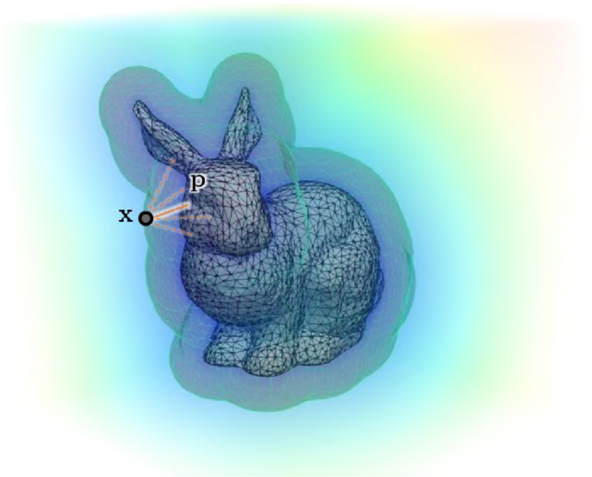


Figure 1.31: Distance function d^+ to a mesh with an isosurface. When computed using the brute force approach we need to compare distances to all mesh triangles for each sampled point $\mathbf{x}$.

Let Γ be a polygonal 2-mesh (see Definition 1.1.8) and $\overline{\Gamma} \subset \mathbb{E}^3$ its union. The *distance function* to set $\overline{\Gamma}$ is defined as:

$$d^+ : \mathbb{E}^3 \to \mathbb{R}^+ : \ \mathbf{x} \mapsto \inf_{\mathbf{p} \in \overline{\Gamma}} \left\{ \|\mathbf{p} - \mathbf{x}\| \ : \ \mathbf{x} \in \mathbb{E}^3 \right\}$$

and its essential modification is

$$d^\pm : \mathbb{E}^3 \to \mathbb{R} : \ \mathbf{x} \mapsto \mathrm{sgn}_{\overline{\Gamma}}(\mathbf{x}) \inf_{\mathbf{p} \in \overline{\Gamma}} \left\{ \|\mathbf{p} - \mathbf{x}\| \ : \ \mathbf{x} \in \mathbb{E}^3 \right\}$$

with the *sign function*:

$$\mathrm{sgn}_{\overline{\Gamma}}(\mathbf{x}) = \begin{cases} 1, & \text{for } \ \mathbf{x} \in \mathrm{Int}(\overline{\Gamma}), \\ -1, & \text{for } \ \mathbf{x} \in \mathrm{Ext}(\overline{\Gamma}), \end{cases} \tag{1.14}$$

[16]Our particular application from Chapter 2 uses a continuous collision detection via a *distance field*.

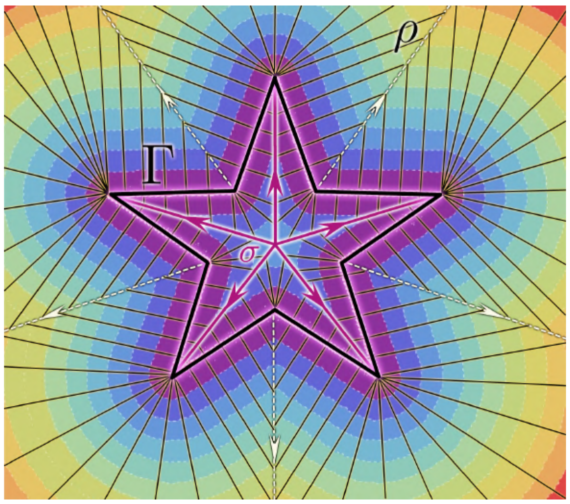


Figure 1.32: A 2D contour plot of (unsigned) distance d^+ to a generating set Γ, showing rarefaction fronts ρ and skeleton σ along with characteristics of d^+ emanating from Γ.

where $\mathrm{Int}(\overline{\Gamma})$ and $\mathrm{Ext}(\overline{\Gamma})$ are the interior and exterior of a watertight surface mesh $\overline{\Gamma}$ (see Section 1.1.6). $d^{\pm}$ is then called the *signed distance function* (SDF). We may also use Γ interchangeably with $\overline{\Gamma}$ if the *generating set* of the distance field is given as an implicit or parametric curve or surface.

Since distance increases linearly away from each point $\mathbf{p} \in \Gamma$, giving rise to linear *characteristics*[17]. The interior of the generating set also contains a bundle of curves/surfaces σ which we call the *skeleton* of Γ. It is a subset of $\mathrm{Int}(\Gamma)$ with a C^1-discontinuity in both the signed $d^{\pm}$ and unsigned d^+ field. Region σ is sometimes also referred to as the *medial axis* of the shape. Due to the properties of Euclidean distance metric, points on Γ itself also have a C^1-discontinuity for the unsigned field d^+. If Γ is concave, the fields will also produce *rarefaction front regions* ρ where the fronts of characteristics spreading outward from Γ meet. This is also where the field becomes C^1-discontinuous (see Fig. 1.32). Converting from d^+ to $d^{\pm}$ reduces the C^1-discontinuity at the generating set Γ.

The complexity of generating set Γ (or union of a mesh $\overline{\Gamma}$) is, of course, imprinted on the field itself. For this reason, we sample d^+ or $d^{\pm}$ as a function on a discrete grid. Now let $\overline{G}$ be a regular grid with $100 \times 100 \times 100 = 10^6$ voxels and let Γ have 5×10^4 triangles, as it is in case of the Stanford bunny model in Fig.1.31. Sampling d^+ for all 10^6 grid nodes requires us to make 5×10^{10} calculations in total for what will be referred to as the *brute force approach*, taking several minutes even when computed on multiple threads.

Accelerating the, so called, *distance query* for meshes has been a standard technique used in many applications, especially game development. Sanchez et al. [60], for example, test multiple possibilities comparing their computational efficiency. As mentioned in Section 1.1.6, Bærentzen and Aanæs published the *angle-weighted pseudonormal* approach in their 2005 paper [14]. Later on, Bærentzen also co-authored an overview of the available methods for SDF computation with colleagues Jones and Sramek [39].

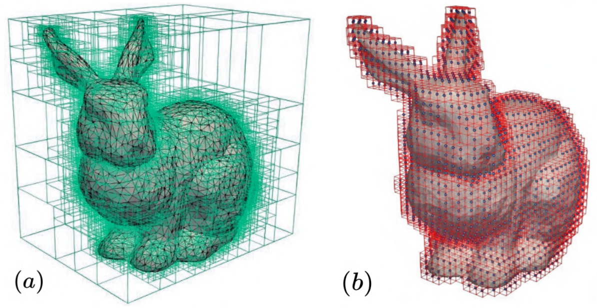


Figure 1.33: The *node boxes* of an *AABB tree* (a), and the *voxel outline* (b) of a simplified *Stanford Bunny* mesh with $N_T = 5002$ triangles and voxel cell size $c_{\overline{G}} = 0.01$.

Bounding Volume Hierarchies

Distance and intersection queries need to be performed on individual mesh primitives such as vertices, edges, or triangles. Hence, we extract a *triangle soup* $\mathcal{T}_\Gamma = \{T_0, ..., T_{N_T}\}$ from mesh Γ. By discarding all connectivity information we can then extend the set of possible input meshes Γ from non-watertight to non-manifold. We prepare a large-enough computational domain G of distance field d which contains $\overline{\Gamma}$. For our applications G is an axis-aligned box expanded from the bounding box B_Γ by some offset $o > 0$. The following algorithm

[17]Curves $\gamma : t \mapsto \gamma(t) \in G$ (in the case of d) constant along the direction of the gradient $\nabla d(\gamma(t)) \cdot \gamma'(t) = 1$.

then outlines the subsequent steps:

```
Algorithm 1: Computing SDF of input mesh $\Gamma$
  Data: A mesh $\Gamma$ with extractable triangle soup $\mathcal{T}_\Gamma$, offset value $o$.
  Result: A set of (signed) distance values $d_{i,j,k}$ sampled over regular grid $\overline{G}$.
1 $\mathbb{T}_\Gamma \leftarrow$ generate an AABB tree from $\mathcal{T}_\Gamma$;
2 generate $\mathbb{O}_\Gamma$ given minimum cell size $c_{\min,\overline{G}} > 0$;
3 create grid $\overline{G} \subset \mathbb{R}^3$ with dimensions of the bounding box of $\Gamma$ and given cell resolution;
4 expand $\overline{G}$ by given offset $o$;
5 set exact distance values $d^{\text{exact}}_{i,j,k}$ to grid points $\mathbf{g}^\Gamma_{i,j,k} \in \overline{G}$ that are centroids of octree $\mathbb{O}_\Gamma$'s leaf cells;
6 set $d_{i,j,k} \leftarrow \infty$ everywhere else;
7 FastSweep($\overline{G}, d_{i,j,k}$);
8 compute sign of $(\overline{G}, d_{i,j,k})$ using voxel flood fill;
```

The *AABB*[18] *tree* $\mathbb{T}_\Gamma$ (from line 1) is a binary bounding-volume tree for the selected collection $\mathcal{T}_\Gamma$ of geometric primitives. The *nodes* of $\mathbb{T}_\Gamma$ are axis-aligned bounding boxes $B = [b_{\min,x}, b_{\max,x}] \times [b_{\min,y}, b_{\max,y}] \times [b_{\min,z}, b_{\max,z}]$ which both contain and intersect their respective subset of the triangle soup (see Fig. 1.33 (a) for illustration). Our AABB tree contains references to actual triangle data only at *leaf nodes.* Searching from the *root node* down towards a leaf has mean complexity $O(\log N_T)$, where $N_T = |\mathcal{T}_\Gamma|$ is the triangle count. Once a leaf is reached, only a minimal amount of geometric primitives (triangles) needs to be iterated through.

```
Algorithm 2: Mesh Distance-Octree
   Data: An AABB tree $\mathbb{T}_\Gamma$ of mesh $\Gamma$
   Result: An octree $\mathbb{O}_\Gamma$ with leaves forming an outline of mesh $\Gamma$.
1  $\mathcal{C}_0 \leftarrow$generate a bounding cube around $\Gamma$;
2  set $\mathcal{C}_0$ as the cube of the root node $\mathcal{O}_{N,0}$;
3  if $\mathcal{C}_0$.size() $> c_{min,\overline{G}}$ then
4      subdivide $\mathcal{C}_0$ into 8 subcells $\mathcal{C}_k,\ k = 1, ..., 8$;
5      foreach $\mathcal{C}_k,\ k = 1, ..., 8$ do
6          if $\mathcal{C}_k$ intersects mesh $\Gamma$ using $\mathbb{T}_\Gamma$ then
7              repeat from line 3 with $\mathcal{C}_0 = \mathcal{C}_k$;
8          else
9              discard $\mathcal{C}_k$;
10         end
11     end
12     break if max depth is reached;
13 else
14     $d^2_{min} \leftarrow \min\{$ squared distance of the centroid of $\mathcal{C}_0$ to $T \in$
       $\mathbb{T}_\Gamma$.GetIntersectingTrianglesWith($\mathcal{C}_0$)$\}$;
15 end
```

We construct and search $\mathbb{T}_\Gamma$ according to the *kD-tree* algorithm in Section 5.2 of de Berg et al. [21]. The most computationally demanding step in the construction of $\mathbb{T}_\Gamma$ is the search for *optimal split position* along

[18]Abbrev. for axis-aligned bounding box. But in some literature it is referred to as a *kD-tree* (k-dimensional).

the longest axis of each non-leaf node's box B. We use the *adaptive resampling* approach according to Hunt et al. [34].

From line 2 in Algorithm 1, $\mathbb{O}_\Gamma$ is an *octree* of mesh Γ constructed according to Algorithm 2. Our octree $\mathbb{O}_\Gamma$ is constructed only for a single purpose, that is: sampling exact distance values $d^{\text{exact}}_{i,j,k} = d_{min}$ (see line 14 in Algorithm 2) on a subset of grid points $\mathbf{g}^{\text{exact}}_{i,j,k} \in \overline{G}$, where $\overline{G}$ is the voxel grid where the distance values (computed in Algorithm 1) are stored. Our spatial alignment corresponds to the fact that the grid points $\mathbf{g}^{\text{exact}}_{i,j,k}$ are centroids of leaf node cubes $\mathcal{C}_l$ of $\mathbb{O}_\Gamma$. The result is a *voxel outline* of mesh Γ in $\mathbb{E}^3$ (see Fig. 1.33 (b)).

Processing the Voxel Field

We know what the desired target voxel size c_G of the resulting grid $\overline{G}$ will be (as an input parameter). Now we need to estimate the initial bounding cell (cube) $\mathcal{C}_0$ (line 1 in Algorithm 2). Since octrees have at most 8 children for each of their nodes, we can compute the proper dimensions and position of the cube of the root node $\mathcal{C}_0$. Let $(\beta_x, \beta_y, \beta_z)^\top = \boldsymbol{\beta} = \mathbf{b}^o_{\Gamma,\max} - \mathbf{b}^o_{\Gamma,\min}$ be the size vector of the *start box* B^o_Γ (bounding box B_Γ of Γ expanded by offset $o > 0$). Then we choose $\beta_{\max} = \max(\{\beta_x, \beta_y, \beta_z\})$ and compute the *expected depth* of octree $\mathbb{O}_\Gamma$:

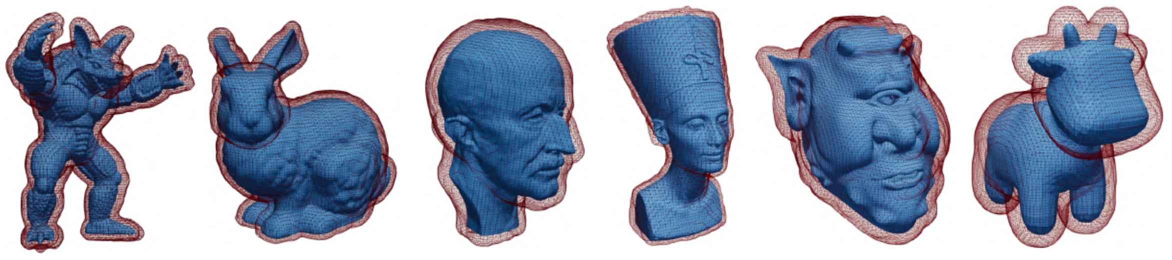
$$D_e = \left\lfloor \log_2 \left(\frac{\beta_{\max}}{c_G} \right) \right\rfloor \ , \quad c_G > 0 \ , \quad \beta_{\max} > 0.$$

The half-size of the octree's root cube will then be $s^0_{1/2} = c_G 2^{D_e}$. For the root center of the root cube, we choose the closest global coordinate

$$\left\lfloor \frac{c_i}{c_G} \right\rceil \times c_G \ , \qquad \text{for } i \in \{x, y, z\},$$

where $\lfloor\cdot\rceil$ denotes round operation, and $\mathbf{c}_\Gamma = (c_x, c_y, c_z)^\top = \frac{1}{2}(\mathbf{b}_{\Gamma,\max} + \mathbf{b}_{\Gamma,\min})$ is the center of B_Γ and B^o_Γ.

Furthermore, we initialize grid $\overline{G}$ with numerical infinities everywhere (see line 8 in Algorithm 1), and compute exact distances $d^{\text{exact}}_{i,j,k} < \infty$ at the grid points $\mathbf{g}^{\Gamma}_{i,j,k}$ *activated by the intersection with* $\overline{\Gamma}$. The intersection query is, of course, accelerated with the help of AABB tree $\mathbb{T}_\Gamma$.



Armadillo	*Bunny*	*MaxPlanck*	*Nefertiti*	*Ogre*	*Spot*
(Stanford)	(Stanford)	(Leif Kobbelt)	(Cosmo Wenman)	(Keenan Crane)	(Keenan Crane)
$N_V = 172\,974$	$N_V = 35\,947$	$N_V = 50\,077$	$N_V = 1\,009\,118$	$N_V = 19\,985$	$N_V = 2\,930$
$N_T = 345\,944$	$N_T = 69\,451$	$N_T = 99\,991$	$N_T = 2\,018\,232$	$N_T = 39\,856$	$N_T = 5\,856$
~9 s	~9 s	~8 s	~12 s	~10 s	~8 s

Figure 1.34: Contours of signed distance fields of six test meshes with N_V vertices, N_T triangles, and average computation times for grid $\overline{G}$ of resolution approx. 250^3 voxels on AMD Ryzen 7 3800x.

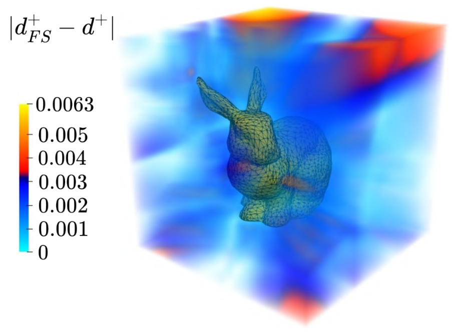


Figure 1.35: Absolute error of the fast-sweeping method compared to brute force ground truth.

The computation of the remaining values follows by using the *fast-sweeping algorithm* introduced by Zhao [77] (see procedure `FastSweep` on line 7 in Algorithm 1). This technique relies on the fact that d^+ is a solution to a non-linear hyperbolic *eikonal equation*: $\|\nabla d^+\| = 1$, and that the field d^+ can be updated from $2^3 = 8$ different directions along the characteristics of d^+ (see Fig 1.32). Field d^+_{FS} computed by the fast-sweeping technique, of course, has some error which accumulates mainly around the corners of the field grid (see Fig. 1.35).

Flood Fill and Pseudo-Watertightness

The *flood fill* algorithm (on line 8 of Algorithm 1, used, for example, by Huska et al. [37]) recursively fills the grid with seeded values by checking for processed neighboring voxels. As mentioned previously, voxels in the outline produced by octree $\mathbb{O}_\Gamma$ are frozen, that is: they cannot be processed by the flood fill. We first negate the grid values $d_{i,j,k} \leftarrow (-d_{i,j,k})$ everywhere on $\overline{G}$, and then find the first exterior voxel that is not frozen (i.e.: is not a part of the voxel outline). Then the recursive voxel flood spreads through neighboring voxels switching the signs of the values back to positive. If the outline voxels form a watertight shape, the interior voxels $C_G \subset \text{Int}(\overline{\Gamma})$ keep negative sign in the distance values $d^{\pm}_{i,j,k} < 0$. This happens to be the case when the holes in $\overline{\Gamma}$ are smaller than voxel size c_G. If so, we may refer to $\overline{\Gamma}$ as a *pseudo-watertight* mesh.

Discussion

The performance measurements for our approach outlined in Algorithm 1 are presented in Fig. 1.34, Fig. 4.3, and in [5] (with full comparison of specific subroutines in Fig. 9), and for 3 different CPUs with additional settings. We also tested this approach against the pseudonormal technique by Bærentzen and Aanæs [14] without *oriented bounding boxes* and *ray casting*, sampling every grid point $\mathbf{g}_{i,j,k} \in \overline{G}$. The available nearest primitive query `pmp::TriangleKdTree::nearest` [69] carries significant overhead for interpolating actual nearest point for each triangle. Therefore our approach is nearly 200 times faster because it exploits locality of the voxelization (see line 5 in Algorithm 1), but also of the fast-sweeping algorithm [77] which is $O(|\overline{G}|)$. Due to the requirement for stack containers, the flood fill step remains to be the largest bottleneck in the whole procedure.

1.3.3 Other Conversion Methods

The remaining forms of conversion between the main geometry representations outlined in Fig. 1.1 are inherently associated with the functional representation. Although not used in the implementation of our application from Chapter 2, they might find their use in our future work, and are thus worthy of brief mention.

A voxel field from an implicit function defined on a simply connected domain $G \subset \mathbb{E}^3$ can be easily obtained by simply sampling the values $f_{i,j,k} \leftarrow f(\mathbf{g}_{i,j,k})$ on grid points $\mathbf{g}_{i,j,k} \in \overline{G}$. The strategy for the opposite direction, on the other hand, requires an approach similar to converting mesh data to implicit functions, namely fitting some *basis functions* to the input spatial data.

While it may seem questionable whether we may be required to convert values at a voxel grid to an FRep, one might imagine that in some future work, this option may well be considered, for example, to construct implicit representations directly from image data. As mentioned above, fitting a set of functions onto a dataset of mesh vertices [76, 47, 78].

Chapter 2

Lagrangian Shrink-Wrapping

This chapter elaborates on the practical results achieved using a special technique for converting a voxel or functional representation to a mesh surface. Alternatively, a mesh input can first be converted to implicit or image data propagating its shape into the surrounding space, and then processed in such fashion.

In particular, this approach involves wrapping a mesh around an implicit or image representation of a shape, and then processing the mesh to refine its shape and capture important details. The result is a highly accurate and detailed representation of the original shape, which can be used in a variety of applications, such as remeshing [44], segmentation [54], extraction of surface representation from point cloud input [20], and the removal of hidden surface components to reduce complexity of the wrapped model [36]. In this chapter, we will examine the underlying principles of Lagrangian shrink-wrapping, including the mathematical foundations of the model. We will also explore the practical results achieved using this technique, highlighting its strengths and limitations, demonstrating its potential in various real-world scenarios.

2.1 Lagrangian Surface Evolution

Analogously to an approach for simulating particle movement in fluid dynamics, the name *Lagrangian* refers to modeling the points of a single contour or a surface rather than an entire field in $\mathbb{E}^3$. We will start with a particular starting surface, for example, a sphere encapsulating the target shape in its interior, and let the surface undergo evolution.

Definition 2.1.1. (Surface Evolution) Let X be a Riemannian 2-manifold. Then for time interval $[0, T]$ a smooth map $F : [0, T] \times X \to \mathbb{E}^3$ is called *surface evolution* in $\mathbb{E}^3$ if $F^t = F(t, \cdot)$ is an immersion of X into $\mathbb{E}^3$ for all $t \in [0, T]$.

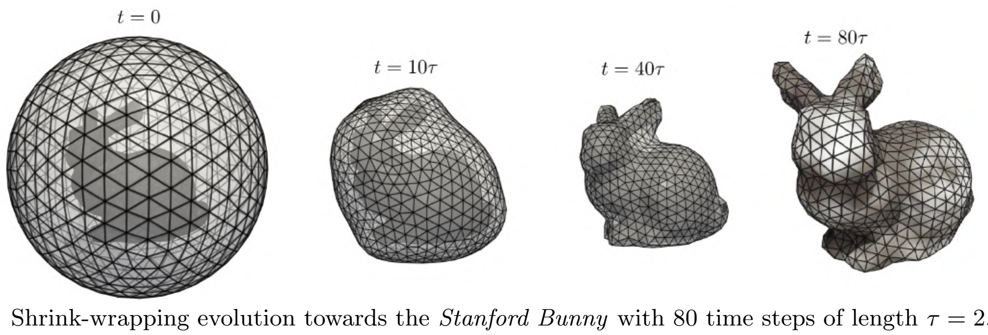


Figure 2.1: Shrink-wrapping evolution towards the *Stanford Bunny* with 80 time steps of length $\tau = 2.5 \times 10^{-3}$.

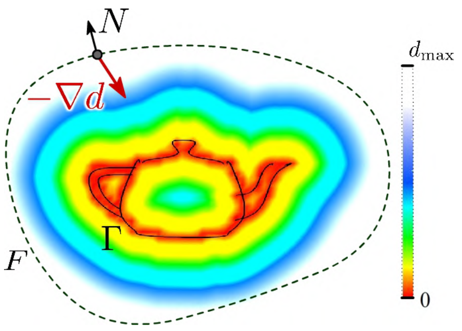


Figure 2.2: Slice of distance field d to surface Γ *Utah Teapot* with resolution 120^3 and an evolving surface F driven by fields d and $-\nabla d$.

According to [54], the velocity of the immersion $\partial_t F = v$ can be decomposed as:

$$\partial_t F = v_N + v_T, \tag{2.1}$$

where v_N and v_T are the normal and tangential components respectively. Such decomposition is guaranteed by the condition that F^t is an immersion of X into $\mathbb{E}^3$. Equation (2.1) then needs to be accompanied by an initial immersion $F^0 = F(0, \cdot)$.

Throughout this chapter, we shall use the map F^t and its image $\mathrm{Im}(F^t)$ for $t \in [0, T]$ interchangeably, and also if particular time t is overlooked, we simply use F, referring to the evolving surface at any time.

Fairing models, using k-th order Laplacian smoothing can be formulated as $\partial_t F = \Delta^k_{g_F} F$ (for more details see Sections 4.2 and 4.3 in [11]) with the given initial condition. Δ_{g_F} denotes the *Laplace-Beltrami operator* on the surface with respect to the metric g_F of its immersion F. For $k = 1$ we have

$$\partial_t F = \Delta_{g_F} F = -HN \ , \quad F(0, \cdot) = F^0, \tag{2.2}$$

where $H = \kappa_{\min} + \kappa_{\max}$ is the mean curvature with principal curvatures $\kappa_{\min}$ and $\kappa_{\max}$, and N an outward-pointing unit normal to the image of F.

First-order Laplacian smoothing is also known as *mean curvature flow* (MCF). Its only known compact exact solution in $\mathbb{E}^3$ is the shrinking sphere: $r(t) = \sqrt{r_0^2 - 4t}$ with starting radius r_0 which solves an ordinary differential equation:

$$r'(t) = -\frac{2}{r(t)}, \ r(0) = r_0.$$

Now let $\Gamma \subset \mathbb{E}^3$ be a *target set*, potentially a non-manifold surface (see Fig.2.2). Let $d : \mathbb{E}^3 \to \mathbb{R}$ be the distance field of Γ, taking either the unsigned d^+ or signed $d^\pm$ form, the latter of which distinguishes the interior $\mathrm{Int}(\Gamma)$ from the exterior $\mathrm{Ext}(\Gamma)$ of Γ with negative and positive sign respectively.

The target set Γ generates its distance field d in ambient space $\mathbb{E}^3$, and for this reason, it can affect the evolution of F in the following advection-diffusion model:

$$\partial_t F = \epsilon \Delta_{g_F} F + \eta N + \rho v_T \ , \quad F(0, \cdot) = F^0, \tag{2.3}$$

where ϵ, η are control functions for the two main components of evolution in the normal direction (summing up to velocity v_N in (2.1)). In our experiments we choose $\rho = \epsilon$. Using control functions:

$$\epsilon(d) := C_1\big(1 - \mathrm{e}^{-d^2/C_2}\big) \ , \quad C_1, C_2 > 0, \tag{2.4}$$

$$\eta(d) := D_1 d\big((-\nabla d \cdot N) - D_2\sqrt{1 - (\nabla d \cdot N)^2}\big),$$
$$D_1 > 0, \ D_2 \geq 0, \tag{2.5}$$

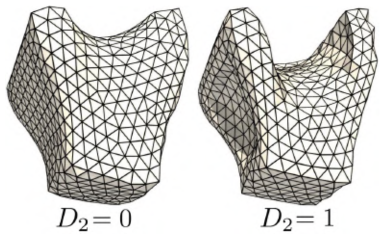


Figure 2.3: The difference between surface F evolving towards *Bent Chair* mesh (see Fig. 3 in [5]) at 55th step with different values of constant D_2.

inspired by [37] using the distance field d to target Γ. With ϵ we ensure that MCF slows down to zero as F approaches Γ, and η controls the orientation of unit normal N to surface F (see Fig. 2.3). To make sure that F does not evolve past Γ under the influence of MCF, we can use modified weight functions ϵ^+ and η^+ which are non-zero only for positive values $d > 0$.

Finally, the tangential movement of surface points with velocity v_T is justified, according to [54], by the fact that for simple MCF (2.2), point density tends to accumulate in areas with high curvature. For this reason Mikula et al. [54] propose two methods of *tangential redistribution*, namely *area-based* and *length-based*. Huska et al. [37] introduce a more lightweight form of *angle-based* redistribution which homogenizes angles of polygons at each mesh vertex. We chose the latter to help with point redistribution in accordance with further adaptive remeshing technique applied in the finishing steps of the evolution of a discretized surface F.

2.2 Discrete Model

2.2.1 Scalar and Vector Fields

Scalar and vector fields (d and $-\nabla d$) are sampled on regular voxel grids G with cell size $c_G > 0$. We start with an axis-aligned bounding box B_Γ of Γ. Given an expansion offset $o_\varsigma = \varsigma\beta_{\min},\ \beta_{\min} = \min(\{\beta_x, \beta_y, \beta_z\})$ with factor $\varsigma > 0$ and size vector $\boldsymbol{\beta} = \mathbf{b}_{\max} - \mathbf{b}_{\min}$ of bounding box B_Γ with min and max points $\mathbf{b}_{\min}, \mathbf{b}_{\max}$, we define the field values on a voxel grid $G \subset \mathbb{R}^3$ using expanded bounds B_{Γ, o_ς} (see Fig. 2.5). We use the *global grid* approach from Section 1.2.2. d is computed using the technique from Section 1.3.2. The gradient field $\nabla d : \overline{G} \to \mathbb{R}^3$ is computed using *central difference*:

$$\nabla d_{i,j,k} = \frac{1}{2c_G}\begin{pmatrix} d_{i+1,j,k} - d_{i-1,j,k} \\ d_{i,j+1,k} - d_{i,j-1,k} \\ d_{i,j,k+1} - d_{i,j,k-1} \end{pmatrix},\ i = 1, ..., N_x - 2,\ j = 1, ..., N_y - 2,\ k = 1, ..., N_z - 2,$$

and, of course, normalized to ensure $\nabla d_{i,j,k}/\|\nabla d_{i,j,k}\| = \widehat{\nabla d_{i,j,k}} \in \mathbb{S}^2$, satisfying the eikonal equation $\|\nabla d\| = 1$ on G.

2.2.2 Mesh Surface

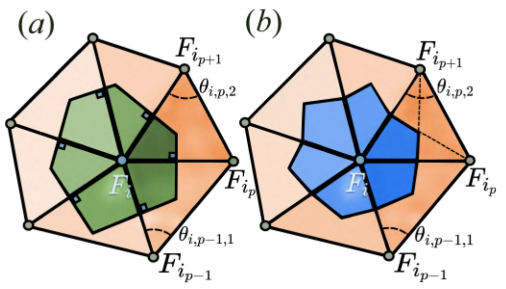


Figure 2.4: Two definitions of a *co-volume* V_i around a vertex F_i of a triangular mesh. For (a) the boundary vertices of V_i in the interiors are line segments of a *Voronoi* region bound to the surface, and in (b) the barycenters of triangles form boundary ∂V_i.

The evolving surface F is approximated as a watertight 2-manifold triangle mesh with vertex set $V = \{F_1, ..., F_{N_V}\}$. Since we need to extend our notation to $3N_V$-dimensional space of the resulting linear system (2.7), as we mentioned in Section 2.1, instead of using bold symbols $\mathbf{f}_i$ for mesh vertices (according to Chapter 1), we refer to points of the surface using the same symbol as that of the immersion F to distinguish the points in $\mathbb{E}^3$ from those in $\mathbb{R}^{3N_V}$.

The starting surface F^0 in our experiments will be an icosahedral tessellation of a sphere (see Section 1.1.5) with subdivision level $s > 0$ encapsulating the target set Γ. An i-th vertex of a geometric realization of the surface at time $t = n\tau \in [0, T],\ n \in \{0, ..., N_T\}$ will be denoted as F_i^t. Value $\tau > 0$ will be referred to as the *length of a time step*.

Equation (2.3) modeling the evolving surface is discretized using finite 2-volumes (areas) surrounding vertices. Co-volumes tessellate the union of the surface $\overline{F^t}$ forming a dual mesh to the pre-existing triangulation. The concept of *co-volumes* (see Fig.2.4) and their use in expressing differential operators on surface meshes is described in more detail by Meyer et al. [52]. Mikula et al. [54] use them to derive the balance of curvature flow between mesh vertices, which converges to the smooth model of MCF with finer tessellations.

Let $V_i \subset \overline{F^t}$ such that $F_i^t \in V_i$ be a co-volume around vertex F_i^t. Then we proceed by integrating (2.3) over each co-volume:

$$\iint_{V_i} \frac{F_i^{t+\tau} - F_i^t}{\tau} \mathrm{d}\mu_{g_F^t} = \iint_{V_i} \epsilon_i^t \Delta_{g_F^t} F_i^{t+\tau} \mathrm{d}\mu_{g_F^t} + \iint_{V_i} (\eta_i^t N_i^t + \rho_i^t v_{T,i}^t) \mathrm{d}\mu_{g_F^t}. \tag{2.6}$$

The Laplace-Beltrami term in (2.3) is discretized implicitly[1] using cotangent scheme

$$\iint_{V_i} \Delta_{g_F} F \, \mathrm{d}\mu_{g_F} \approx \frac{1}{2} \sum_{p=1}^{m} \big(\cot \theta_{i,p-1,1} + \cot \theta_{i,p,2} \big)(F_{i_p} - F_i)$$

(derived in Appendix A of Mayer et al. [52]) using cotangents of angles $\theta_{i,p-1,1}$ and $\theta_{i,p,2}$ opposing to each edge $e^i_p = F_i F_{i_p} \in \mathrm{St}(\{F_i\})$ from central vertex (see Fig. 2.4). The integral equation (2.6) transforms into a system:

$$\underbrace{\mathrm{A}^t_{ii} F^{t+\tau}_i + \sum_{p=1}^{m} \mathrm{A}^t_{ii_p} F^{t+\tau}_{i_p}}_{(\mathbf{A}^t \boldsymbol{F}^{t+\tau})_i} = \underbrace{F^t_i + \tau(\eta^t_i N^t_i + \rho^t_i v^t_{T,i})}_{(\mathbf{b}^t)_i},$$

for $i = 1, ..., N_V$, where m is the valence of vertex F_i, with matrix coefficients:

$$\mathrm{A}^t_{ii} = \left(1 + \frac{\tau \epsilon^t_i}{2\mu_{g^t_F}(V_i)} \sum_{p=1}^{m} \big(\cot \theta^t_{i,p,1} + \cot \theta^t_{i,p,2} \big) \right),$$

$$\mathrm{A}^t_{ii_p} = -\frac{\tau \epsilon^t_i}{2\mu_{g^t_F}(V_i)} \big(\cot \theta^t_{i,p-1,1} + \cot \theta^t_{i,p,2} \big),$$

where $\mu_{g^t_F}(V_i)$ are 2-dimensional Lebesgue measures of co-volumes V_i, and values ϵ^t_i and η^t_i are computed with (2.4) from trilinearly-interpolated distance values d^t_i at positions of vertices F^t_i. The matrix $\mathbf{A}^t$ of the resulting system

$$\mathbf{A}^t \boldsymbol{F}^{t+\tau} = \mathbf{b}^t, \tag{2.7}$$

is diagonally-dominant and sparse. To impose Dirichlet boundary conditions on ∂F, we simply put $A^t_{ii} = 1$, $A^t_{ii_p} = 0$, and $b^t_i = f(F^t_i)$. If f is an identity, the boundary vertices remain fixed in space.

Optionally, velocity v_T tangent to the surface at each vertex is computed using the angle-based approach:

$$v_{T,i} = \mathrm{proj}_T \left(\frac{\omega}{m} \sum_{p=1}^{m} \left(1 + \frac{e_{i_p}}{\|e_{i_p}\|} \cdot \frac{e_{i_{p+1}}}{\|e_{i_{p+1}}\|} \right) (e_{i_p} + e_{i_{p+1}}) \right),$$

with weight $\omega > 0$, where $e_{i_p} = F_{i_p} - F_i$ and $e_{i_{p+1}} = F_{i_{p+1}} - F_i$ are edge vectors from the central vertex F_i belonging to a single triangle. Projection operator $\mathrm{proj}_T(v) = v - (v \cdot N)N$ with surface normal N projects vectors v to tangent plane of F. Inspired by [37], this movement homogenizes angles of adjacent polygons at vertex F_i.

Mikula et al. [54] also lay the theoretical foundations for a more complicated approach of tangential redistribution, namely the *(asymptotically uniform) 2-volume-based redistribution.* The key insight from this method is that one way to homogenize co-volumes is to assume that the ratio between *volume density* (density of V_i per unit of surface area) and total surface area approaches a constant as $t \to \infty$. This leads to the construction of a (pull-back) vector field determined by the gradient of an unknown *redistribution potential* ψ^t defined on F^t which needs to be evaluated for each time step as a solution of a Poisson problem.

Linear system (2.7) is solved for each vertex F_i and time step. Additional remeshing operations can be applied to increase mesh quality. The method is implicit in the Laplacian term, and explicit in time, therefore we refer to it as a *semi-implicit* formulation.

[1]For time $t + \tau$ and t computed in the previous step.

2.3 Numerical Stability

According to Section 3 in [53], the semi-implicit finite volume approach for curves in $\mathbb{E}^2$ leads to stability constraint $\tau \approx h^2$ where $h > 0$ is a spatial step and $\tau > 0$ a time step. In the co-volume formulation on F this translates to

$$\tau \approx \mu(V), \tag{2.8}$$

where $\mu(V)$ is the 2-dimensional Lebesgue measure of a co-volume V of an arbitrary vertex in F. Theoretically, large deviations from (2.8) should result in the formation of singularities in F. Practically however, the variability of measures $\mu(V)$ in unstructured meshes implies only an approximate control.

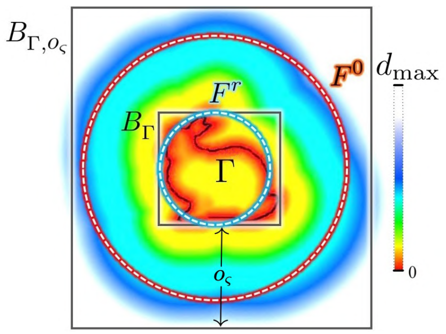


Figure 2.5: Under the stability assumption we consider that an icosahedral tessellation of a sphere F^0 evolves into an *expected sphere* F^r with radius $r > 0$.

In Section 2.4 of [5], we introduced a *scale-based heuristic* for securing numerical stability and simulated the results in Section 4.3. Simply put, when performing time iterations on surface F, we scale it uniformly by factor:

$$\phi = \sqrt{\frac{\tau}{\sigma \mu_r(V)}} \tag{2.9}$$

derived from (2.8) for some *shrink factor* $\sigma > 0$, and apply inverse scaling when we want to transform the result into the original size. Since F takes an a priori unknown shape in the last time step $T = N_t \tau$, we can only approximate the *predicted mean co-volume measure* $\mu_r(V)$ as the mean area of co volumes V of homogeneously distributed vertices on an *expected sphere* F^r with radius $r > 0$. Sphere F^r is an approximation of F at end time T undeformed by the effects of target shape Γ.

In our experiments, we use an empirical estimate

$$r \approx 0.4(\beta_{\min} + (0.5 + \varsigma)\beta_{\max}) \tag{2.10}$$

where $\beta_{\min}$ and $\beta_{\max}$ are the minimum and maximum sizes of target set bounding box B_Γ, and ς is the expansion factor (see Section 2.2.1). The measure estimate for evenly distributed N_V^s vertices on a sphere is:

$$\mu_r(V) = \frac{4\pi r^2}{N_V^s}.$$

where we evaluate vertex count N_V^s of an icosahedral tessellation of a sphere with subdivision level s according to Theorem 1.1.2 with $N_V^0 = 12$, $N_E^0 = 30$.

Different shrink-wrapping schemes, such as the *isosurface evolution* in Section 2.7.2, yield different stability schemes to satisfy criterion (2.8). In particular, if the starting surface F^0 is an isosurface generated by the marching cubes algorithm (see Section 1.3.1), the voxel size $c_G > 0$ of the triangulated grid $\overline{G}$ gives a suitable estimate for the co-volume measure.

The triangle T with the largest area that can fit into a cube voxel with size c_G has area $\frac{\sqrt{3}}{2}c_G^2$. The size of the barycentric co-volume V_{c_G} corresponding to the aforementioned triangle is, of course, $\frac{1}{3}\mu(T)$. Such triangle can then occur in four neighboring voxels sharing a central vertex F^i (see Fig. 2.6 (a)). Therefore we have

$$\mu(V_i) = \frac{2\sqrt{3}}{3}c_G^2, \tag{2.11}$$

which is then just substituted to (2.9) for the co-volume measure.

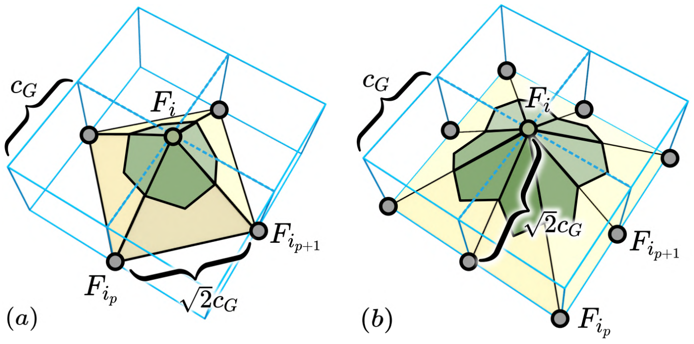


Figure 2.6: Estimated co-volumes constructed by polygonizing a scalar field in four neighboring voxels.

We should note that the configuration in Fig. 2.6 (a) does not maximize the measure. Replacing the maximal equilateral triangle with two right triangles with hypotenuse aligned with the voxel diagonal (Fig. 2.6 (b)) yields larger measure

$$\mu(V_i) = \frac{4\sqrt{2}}{3}c_G^2. \tag{2.12}$$

Another assumption in this case is homogeneity of the distribution of mesh vertices along the starting isosurface F^0, which is certainly not guaranteed by the interpolated positions of vertices at voxel edges. To secure this criterion, we simply run one iteration of adaptive remeshing on starting surface F^0 before starting the evolution.

2.4 Feature Detection

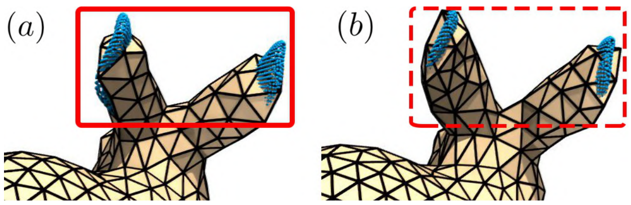


Figure 2.7: A trade-off between local triangle quality and feature detection: Results after using cosine-based (a) and curvature-based (b) feature detection. Points of the target surface (*Stanford Bunny*) are shown in blue.

In mesh processing, features refer to distinctive subsets of the mesh surfaces, such as sharp corners, creases, and boundaries, which carry important information about the shape of the object being represented. Accurately identifying and preserving these features is important for downstream applications, and can simplify processing by providing a suitable level of abstraction.

Feature elements (vertices and edges) require a slightly different treatment during evolution. We need the sensitivity to *true features*[2], and on the other hand avoid marking false positives at *convex-dominant saddle* (CDS) *points* (see Definition 2.4.1).

The most straightforward approach is to mark all edges e with a large-enough dihedral angle α_e. This approach is implemented in the PMP library [69] for cosines[3] $(N_i \cdot N_j) = \cos\alpha_N$ with unit normals N_i and N_j to faces sharing edge e. This, however, leads to the loss of sharp protrusions such as limbs or ears (see Fig.2.7) because the evaluation has only one degree of freedom, favoring sharp edges surrounded by relatively flat patches specific for artificial objects. These do not arise naturally as a consequence of flow (2.3) and are difficult to enforce even manually for organic target shapes.

Since we require an automatic process during surface evolution, we propose a vertex-based detection evaluating the *angle of mean curvature* $\gamma = Hl$ where l is the arc length of smooth surface F. In the discrete

[2]Stemming from the shape of generating set Γ.
[3]The dihedral angle is then $\alpha_e = \alpha_N + \pi/2$, but $\cos\alpha_N$ suffices for the binary feature/non-feature evaluation.

setting we evaluate $\overline{\gamma}_i$ at vertex $\overline{F}_i = F_i$ using *neighborhood mean edge length*:

$$\bar{l} = \frac{1}{m} \sum_{p=1}^{m} \|e_{i_p}\|,$$

also used by Smith [70] to estimate the local radius of curvature for a surface update term. For mean curvature we use the cotangent estimate by Meyer et al. [52]:

$$H_i \approx \frac{1}{4\mu(V_i)} \sum_{p=1}^{m} \left(\cot\theta_{i,p-1,1} + \cot\theta_{i,p,2}\right) \left\|F_{i_p} - F_i\right\|.$$

Using $\overline{\gamma}$ we convert mean curvature H to a scale-invariant quantity. Vertices with $\overline{\gamma} = 2\overline{lH} < \gamma_{\text{crit}}$ are marked as feature. Unfortunately, this alone leads to a decrease of mesh quality in CDS vertices (see Definition 2.4.1).

In our further evaluation, we require principal curvatures:

$$\kappa_{\max,i} = H_i + \sqrt{H_i^2 - K_i}$$
$$\kappa_{\min,i} = H_i - \sqrt{H_i^2 - K_i}$$

where K_i is the *Gaussian curvature estimate*:

$$K_i = \frac{2\pi - \sum_{p=1}^{m}\theta_{i_p}}{\mu(V_i)}, \quad \theta_{i_p} = \arccos\frac{(e_{i_p}\cdot e_{i_{p+1}})}{\|e_{i_p}\|\|e_{i_{p+1}}\|},$$

according to [52].

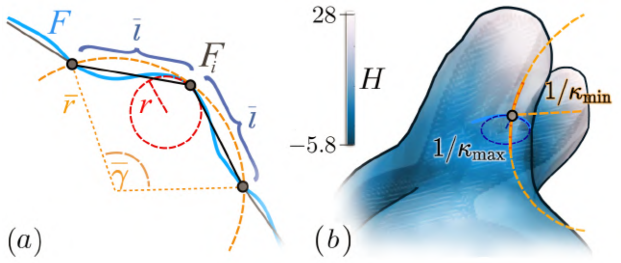


Figure 2.8: (a): An estimate of mean curvature angle $\overline{\gamma}$ at vertex F_i. (b): The imbalance of principal curvatures $\kappa_{\max}$ and $\kappa_{\min}$ at a *convex-dominant saddle vertex*. The color values show mean curvature H.

Definition 2.4.1. (CDS Points) Let the image of F be a 2-manifold surface in $\mathbb{E}^3$ with principal curvatures $\kappa_{\max}$ and $\kappa_{\min}$. Saddle points where

$$|\kappa_{\max}| < K|\kappa_{\min}|, \quad K > 1,$$

are said to be *convex-dominant saddle* (CDS) *points* (see Fig. 2.8 (b)).

CDS vertices can become false positives for feature detection, and thus we propose not to mark them. Additionally, we should also not mark vertices with valence $m > 6$ because the following adaptive remeshing steps will avoid fixing them.

2.5 Adaptive Remeshing

[23] and [10] describe a remeshing framework for triangle meshes based on topological operations such as split, collapse, and flip. It can be repeated $N_{\text{rem}} > 0$ times using the *adaptive technique* which also preserves the positions of edges and vertices marked as feature (see Section 2.4).

Considering scale bounds discussed in Section 2.3, the sizing for remeshing is computed from the edge lengths of the icosahedron:

$$l_{\min} = 2\lambda_{\min} r \sin\left(\gamma_{ico} 2^{-(s+1)}\right), \quad l_{\max} = \lambda_{\max} l_{\min}, \tag{2.13}$$

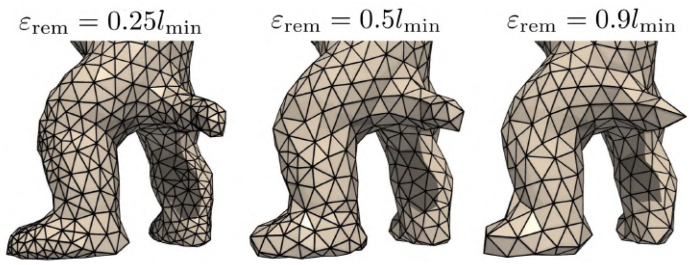

$\varepsilon_{\text{rem}} = 0.25 l_{\min}$
$\varepsilon_{\text{rem}} = 0.5 l_{\min}$
$\varepsilon_{\text{rem}} = 0.9 l_{\min}$

Figure 2.9: The effect of changing approximation error ε_{rem} after 80 time steps for adaptive remeshing as a multiple of minimum edge length $l_{\min} = 0.0799$ of *Armadillo* (in stabilized scale $\phi = 0.0261$).

where $\gamma_{ico} = 2\pi/5$, $\lambda_{\max} > \lambda_{\min} > 0$ are controllable scale factors, and r the radius of (properly scale-adjusted) geodesic icosahedron of subdivision level s.

For the isosurface evolution (see Section 2.7.2) we have

$$l_{\min} = \lambda_{\min} \sqrt{2} c_G, \tag{2.14}$$

where c_G is the cell size of the triangulated scalar grid (see Fig. 2.6 (a) for the preferred equilateral configuration).

Combined with error $\varepsilon_{\text{rem}} = \max\{\|F - \overline{F}\|\}$ between linear approximation $\overline{F}$ and surface arc F, we obtain adaptive sizing values for all edges [23]. For our evolving surface, the sizing needs to be adapted to stabilized scale ϕ. It should also be noted that strong (low-ε_{rem}) adaptivity may adversely affect co-volume sizing necessary for stability (see Fig. 2.9), especially for feature vertices.

2.6 Complete Algorithm and Implementation

At last, we have all the tools necessary to implement the ensuing algorithm of this chapter. In this section, we provide a broader outlook on our implementation, combined with the detailed insights on the particular steps in the shrink-wrapping algorithm and the details of their implementation when necessary.

The full `C++` implementation with the complete list of parameters (which exceeds the scope of this section) can be found in the GitHub repository Implicit Surface Wrap [6] in method `Evolve` of classes `SurfaceEvolver` or `IsoSurfaceEvolver`. In the diagram of our application's key architecture in Fig. 2.10, we separate the computation of signed distance d and surface evolution as two distinct functionalities, even though they are combined in Algorithm 3.

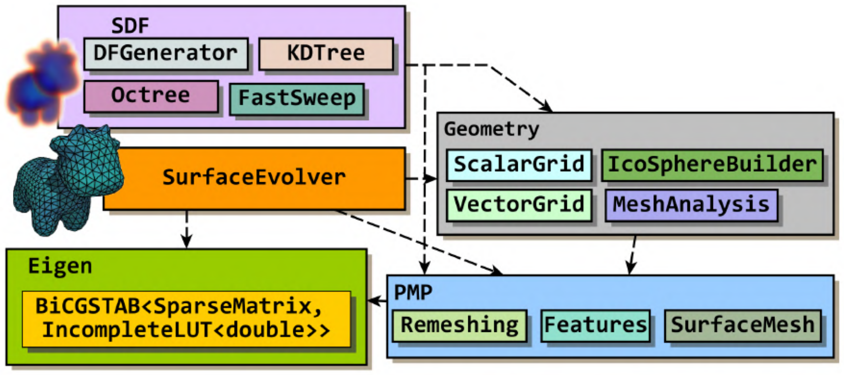


Figure 2.10: The architecture of our application. Dashed arrows denote the *uses* relation.

The main loop of the algorithm will simply perform the update of the evolving surface F, while the surface and the distance field d and its negative gradient $-\nabla d$ will need to be prepared in the preprocessing stage. This stage also incorporates the construction of the starting surface F^0 which can be either a properly centered icosahedral tessellation of a sphere with radius r (computed according to (2.10)), or an isosurface S_{d_0} for a given isolevel $d_0 \in \mathrm{Im}(d^{\pm})$.

Now we proceed to outlining the overall algorithm in pseudocode:

```
Algorithm 3: Shrink-Wrap of a Target Surface $\Gamma$
    Data: A mesh $F^0$ (preferrably of higher quality than $\Gamma$), a target mesh $\Gamma$, time step size $\tau$, number
          of time steps $N_t$, adaptive remeshing parameters $\lambda_{\min}$, $\lambda_{\max}$ etc., offset $o_\varsigma$, tangential
          redistribution weight $\omega$;
    Result: Meshes $F_t$ for each time step, and the result $F_{t_s}$
 1  $(d : \overline{G} \to \mathbb{R}) \leftarrow$ComputeSDF($\Gamma, o_\varsigma$);
 2  $F^0 \leftarrow$BuildStartingSurface($\Gamma, o_\varsigma$);
 3  Estimate co-volume measure $\mu(V)$ from starting surface $F^0$;
 4  $\phi \leftarrow \sqrt{\tau/(\sigma\mu(V))}$; $F^0 \leftarrow \mathbf{M}_\phi F^0$; $\overline{G} \leftarrow \mathbf{M}_\phi \overline{G}$;
 5  compute $-\nabla d$;
 6  compute $l_{\min}$ and $l_{\max}$ (according to (2.13) or (2.14));
 7  for $t = \tau$; $t < t_s = N_t\tau$; $t+ = \tau$ do
 8      ComputeNormals($F^t$);
 9      compose and solve linear system $\mathbf{A}^t \boldsymbol{F}^{t+\tau} = \mathbf{b}^t$ (possibly with tangential redistribution $\mathbf{v}_T \neq \mathbf{0}$);
10      $F^{t+\tau} \leftarrow$UpdateVertices($\boldsymbol{F}^{t+\tau}$);
11      if do remeshing & $t > t_{rem}$ then
12          DetectFeatures($F^{t+\tau}$);
13          $F^{t+\tau} \leftarrow$AdaptiveRemeshing($F^{t+\tau}$, $l_{\min}$, $l_{\max}$, $\varepsilon_{\mathrm{rem}}$);
14      end
15  end
```

The `ComputeSDF` function on the very first line is implemented according to Algorithm 1. With regard to the following step, `BuildStartingSurface` may represent either the construction of an icosahedral tessellation of a sphere encapsulating Γ, or an isosurface of the distance field $d : \overline{G} \to \mathbb{R}$, potentially re-sampled to obtain target discretization. Each vertex of the starting surface F^0 is transformed by matrix $\mathbf{M}_\phi$ with isotropic scaling component ϕ computed according to Section 2.3 depending on the tessellation of F^0. The same transformation needs to be applied to the voxel grid $\overline{G}$ so it aligns with the original position of the surface in the distance field.

For each time step $t = 0, ..., N_t\tau = T$ we need to evaluate unit normals N_i^t for each mesh vertex F_i^t in `ComputeNormals`, and fill the available data into the linear system according to Section 2.2.2. We use the `BiCGSTAB` solver with `IncompleteLUT` preconditioner from the Eigen library[4]. Afterwards, our evolving mesh surface needs to be updated (see line 10 in Algorithm 3), and then post-processed.

We define a time step $t_{\mathrm{rem}} \in \{\tau, 2\tau, ..., T\}$ after which `DetectFeatures` (see Section 2.4), and `AdaptiveReme-shing` (Section 2.5) are called. The reason for this delay is the lack of necessity for the changes in tessellation at the early stages of evolution, especially for the shrinking sphere approach. With the tessellation-changing operations applied on $F^{t+\tau}$ during adaptive remeshing, the amount of vertices N_V of the surface will likely change. Hence, the dimensionality of the linear system at line 9 in Algorithm 3 will require adjustment for each time step.

[4]PMP also uses Eigen internally for matrix and vector representations.

2.7 Results

2.7.1 Shrinking Sphere

First, we verify the stability assumptions outlined in Section 2.3, specifically for the shrink-wrapping with sphere. The *failed* evolutions represent immersions F of mesh which have accumulated so many errors that a substantial portion of their vertices reach beyond the expanded bounds B_{Γ,o_ς} (Section 2.2.1). We can then evaluate stability based on the number of steps until such explosion of points occurs.

The first series of results (without adaptive remeshing) can be seen in Fig. 2.11 for meshes *Bunny* and *Armadillo*. The weight function ρ of angle-based tangential velocity v_T is identically one (left) or $1 - \mathrm{e}^{-d^2}$ (right), while using partial advection term η with $D = 0$. Additional stabilization is achieved by using a distance-dependent weight function ρ.

After introducing adaptive remeshing (Section 2.5) we ensure some level of homogeneity of co-volumes V around vertices of the evolving surface F. Since scaling (2.9) can be weighed by shrink factor $\sigma > 0$, we first assume $\sigma = 1$, and observe the values of co-volume measures $\mu(V)$ for each vertex during $N_t = 80$ steps.

From measures shown, for example, in Fig. 2.12 we deduce $\mu(V) \approx \tau/5$ most of the time,

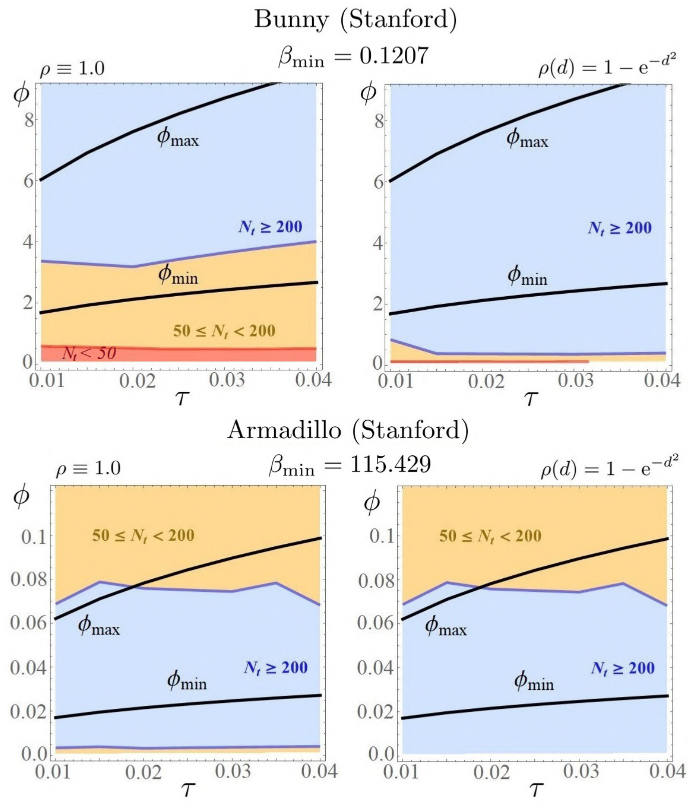


Figure 2.11: Regions of stability (light blue) where evolution completed $N_t \geq 200$ steps without failure. The black curves denoted by $\phi_{\min}$ and $\phi_{\max}$ are the bounds for ϕ by the starting F^0 and the *expected* surface F^r. $\beta_{\min}$ denotes minimum dimension of the bounding box of the target surface Γ. Yellow and red regions correspond to evolutions that failed after $N_t \in [50, 200)$ steps and $N_t < 50$ steps respectively.

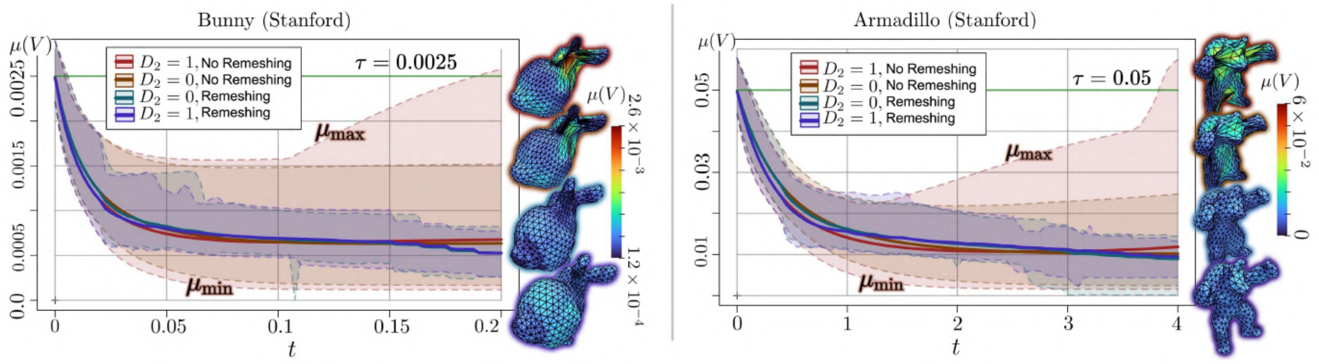


Figure 2.12: The range of co-volume measures $[\mu_{\min}, \mu_{\max}]$ for four different settings with results $F^{80\tau}$ on the right.

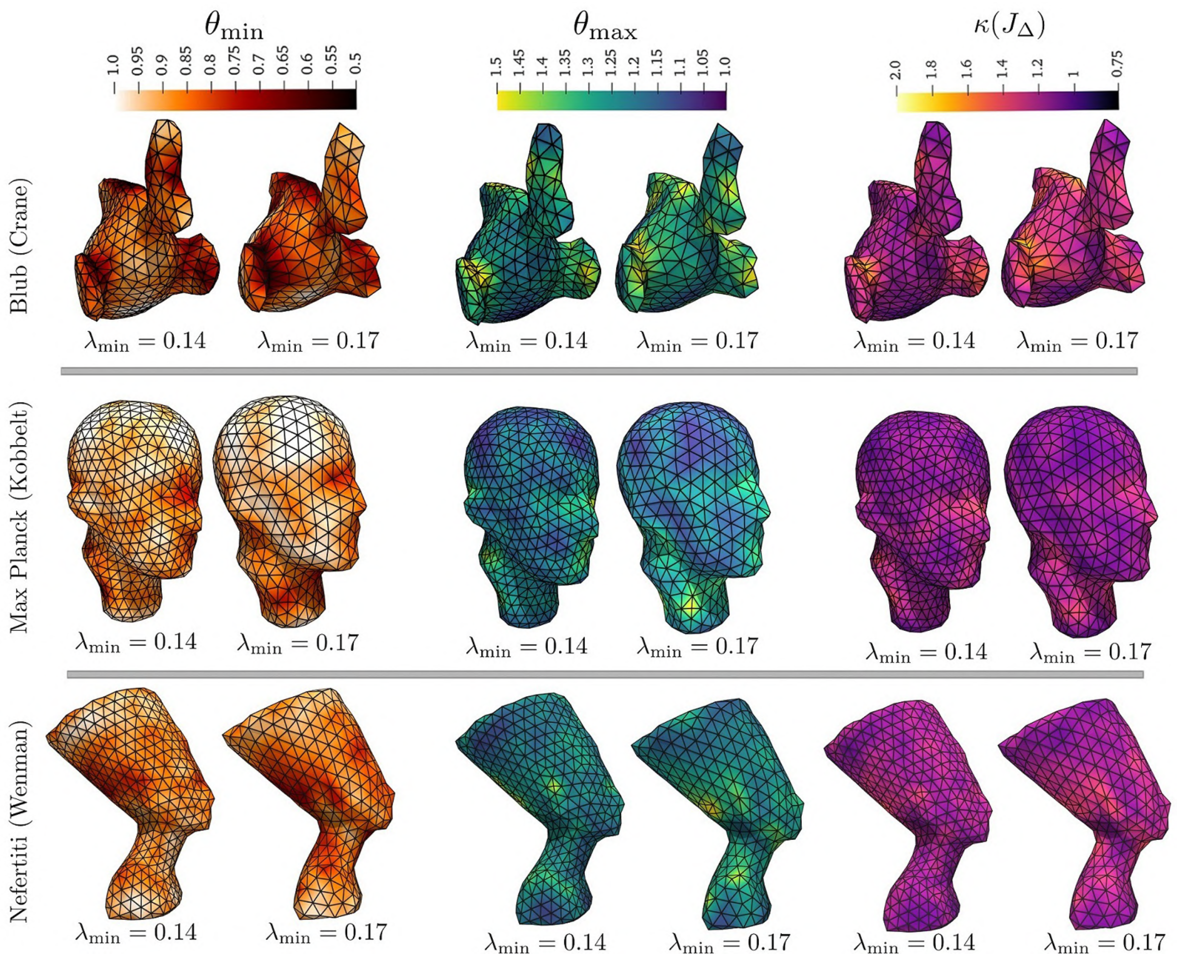


Figure 2.13: Triangle metrics $\theta_{\min}$ (minimum angle), $\theta_{\max}$ (maximum angle), and $\kappa(J_\Delta)$ (condition number of equilateral triangle Jacobian) evaluated for different minimum sizing factors $\lambda_{\min}$.

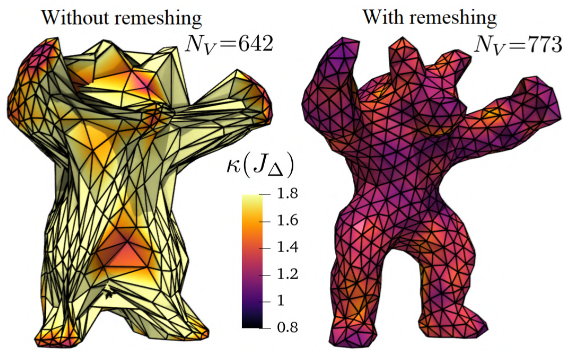


Figure 2.14: A comparison of the quality metric $\kappa(J_\Delta)$ for model (2.3) with $D_2 = 1$ on *Armadillo* mesh with and without remeshing with our feature detection approach with vertex counts N_V.

so we put $\sigma = 1/5$. It is evident that without remeshing, values $\mu(V)$ fluctuate even above the time step size τ, and thus lead to the accumulation of numerical errors.

We proceed by running the evolution model with all of the parameters mentioned above on other meshes, including the evaluation of triangle metrics from Section 1.1.8. A sample of the tests is run for two different values of adaptive sizing $\lambda_{\min} = 0.14$ and 0.17 (see Fig. 2.13). The maximum edge length factor is kept at $\lambda_{\max} = 4$ while the critical angle for feature detection is kept at $\gamma_{\text{crit}} = \pi/2$.

The sizing factor $\lambda_{\min}$ is sensitive to input down to second decimal place. This can be advantageous for second-order sizing, complementing subdivision s of F^0. The use of Voronoi contra barycentric Laplacian (see Fig. 2.4) differs only for some cases in the location of emerging instabilities.

The values of quality metrics in Fig. 2.13 remain mostly within bounds $[\pi/6, \pi/3]$ for $\theta_{\min}$, $[\pi/3, \pi/2]$ for $\theta_{\max}$, and $[1, 1.3]$ for $\kappa(J_\Delta)$. In Fig. 2.14, we notice the major difference in quality metric $\kappa(J_\Delta)$ in an 80-time-step result of our previous implementation [5] and the approach using remeshing with $\lambda_{\min} = 0.14$, $N_{\text{rem}} = 3$, $K = 2$, and using back-projection.

2.7.2 Evolving Isosurface

Icosahedral tessellation of the sphere encapsulating target set Γ cannot change its genus during evolution outlined in Algorithm 3. Therefore we concluded that the most straightforward way to wrap surfaces with genus $g > 0$ is to construct F^0 using a scalar field polygonization technique such as marching cubes (see Section 1.3.1). In this case, we use the specialized stabilization technique mentioned in Section 2.3, namely formulas (2.11) or (2.12).

Choosing the proper isolevel $d_0 \in \text{Im}(d^\pm)$ for signed distance $d^\pm$ on $\overline{G}$ becomes a crucial step when we want to make sure the starting surface evolves towards a shape with the same genus as target Γ. For the purposes of this evaluation we simply measured the interior diameters of individual holes of each input mesh Γ, and estimated the isolevel d_0 for the starting isosurface.

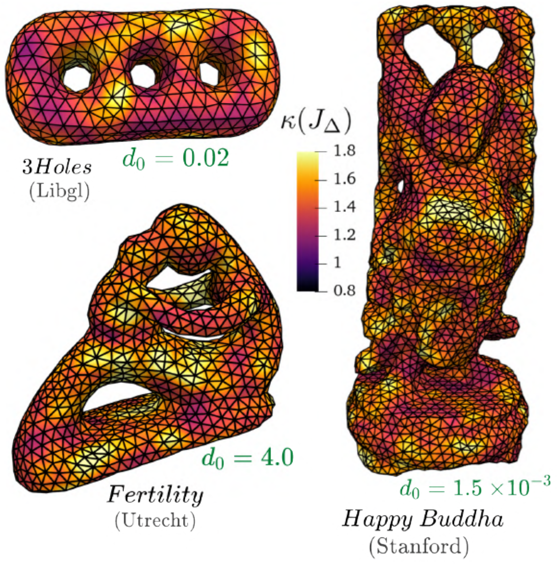


Figure 2.15: Evaluation of quality metric $\kappa(J_\Delta)$ for three target surfaces with higher genus after $N_t = 20$ time steps of size $\tau = 0.05$.

We also tested re-sampling voxel fields $\overline{G} \to \mathbb{R}$ to obtain different level of detail for the evolving mesh. In other words: increasing or decreasing voxel size c_G, and trilinearly interpolating the original grid values, so that the triangles produced by the marching cubes algorithm have a certain mean edge length. The preprocessing, however, still requires at least one step of remeshing to homogenize co-volumes V. Since remeshing provides substantial control of the size of triangles and thus also of the co-volumes, re-sampling is not required. In fact, remeshing accompanies isosurface evolution from its beginning $t = t_{\text{rem}} = 0$.

The more stable version of the tests shown in Figures 2.16 and 2.15 (without substantial fluctuations in $\mu(V)$) used scaling factor ϕ computed with measure estimate (2.12). Throughout the test we also set minimum edge multiplier $\lambda_{\min} = 0.4$ for adaptive remeshing (Section 2.5). Except for a slightly smaller critical mean curvature angle $\gamma_{\text{crit}} = 0.4\pi$ the remaining parameters are the same as for shrinking sphere tests in Section 2.7.1.

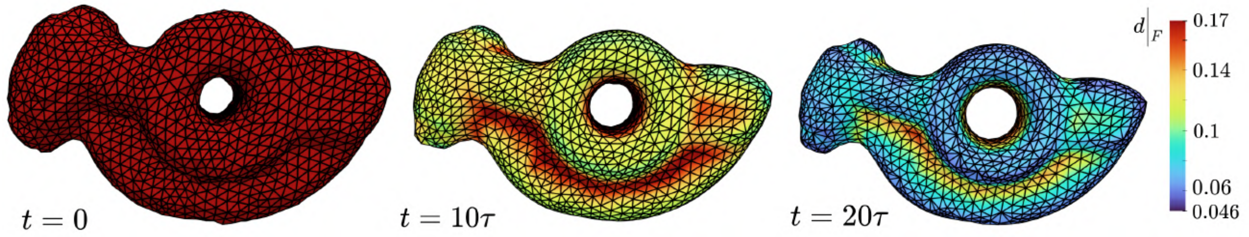


Figure 2.16: Shrink-wrapping of *Rocker Arm* (INRIA) mesh Γ with genus 1 showing values of distance $d|_F$ to Γ interpolated for surface F after 0, 10 and 20 time steps of length $\tau = 0.05$. The starting isosurface was tessellated for isolevel $d_0 = 0.06$.

2.8 Discussion

Lagrangian shrink-wrapping of target sets Γ by an evolving surface F outlined in this chapter is an unconventional method for extracting mesh surfaces from implicit or voxel field representations of solid objects or point cloud data. The semi-implicit finite volume formulation of model (2.3) examined by Mikula et al. [54] and further developed by Huska et al. [37] presented a series of challenges, mainly relating to the numerical stability of the model. We solved the issue of stability for icosahedral tessellations of a sphere (see Fig. 1.18) encapsulating the target set Γ in Cavarga [5]. The stabilization is described in more detail in Section 2.3.

Furthermore, we improved upon our previous work mainly by introducing adaptive remeshing (Section 2.5) and curvature-based feature vertex detection (Section 2.4). On top of using encapsulating spheres as starting surfaces, we extended our approach to surfaces with higher genus by utilizing remeshed isosurfaces computed by the marching cubes algorithm and remeshed during preprocessing.

Although Lagrangian surface evolution models with advection fully wrap a subset of meshes, an extension of this approach to general geometries demands further inquiry. We would also like to point out that only a small sample of possible approaches to tangential redistribution have been tried. The shrink-wrapping implementation would certainly benefit from splitting and/or decimation subroutines for minimal surface regions of the solution F.

Eventually, Algorithm 3 can be extended to quadrilateral meshes, as in [37], with additional analysis of stability for changing density of mesh vertices.

Chapter 3

View and Data-Dependent Triangulation

In Chapter 2, we explored the technique of shrink-wrapping, which allows us to generate a 3D surface that conforms tightly to an object's shape. While this method may be pushed to its limits with high level of detail (LOD) of the output surface, it does not reduce the exterior complexity of the resulting mesh. As we move forward in our discussion, we must consider practical solutions for managing the increasing complexity of geometric data.

In this chapter, we will be building upon the fundamental concepts discussed in Chapter 1, and exploring practical solutions in the context of the latest advancements in geometric conversion, compression of spatial data, and high-performance real-time rendering. As we delve deeper into the world of modern geometry processing, we continue to face the challenge of dealing with large datasets, a problem that has been present since the early stages of this field.

Advancements in hardware performance and the increasing use of parallel processing have made it possible to seamlessly render meshes with millions of polygons. However, as we continue to increase the memory and computational load, achieving real-time response in a user-friendly application becomes more challenging. To address this issue, we must turn to additional techniques for compression and simplification of the input geometry. Balancing the need for high performance with the desire for fidelity and quality of triangulation (see Section 1.1.8) is an ever-present challenge that requires careful consideration.

In Section 1.3, we explored various techniques for geometric conversion, including the use of triangulation to represent 3D surfaces as a mesh of triangles. In this chapter, we will focus on view and data-dependent triangulation of mesh surfaces. By adapting the triangulation of the mesh to the viewpoint of the user or the characteristics of the data being displayed, we can optimize performance while maintaining a high level of fidelity.

Overall, this chapter will provide an in-depth examination of the challenges and solutions related to dealing with large datasets in the context of real-time rendering. We will explore the use of view and data-dependent triangulation as one approach to balancing the need for high performance with the desire for accurate and detailed representation of geometric data.

3.1 Mesh Optimization on Progressive Meshes

Techniques of *mesh optimization*, also referred to as *simplification* or *decimation*, are used to reduce the complexity of polygonal meshes while preserving their visual appearance. One of the most notable results in this area is the development of the *quadric error metric* (QEM) by Garland and Heckbert [24], which provides a measure of the error introduced by mesh simplification and can be used to guide the simplification process. An intriguing approach which we shall elaborate on in this section was introduced by Hugues Hoppe who published a series of results [32, 29, 30] stemming from the concept of *progressive meshes* produced from two atomic types of tessellation-changing operations (see Section 1.1.5), namely: edge collapse Θ and (its inverse) vertex split Θ^{-1}, using a specialized energy functional that can be used to optimize the connectivity of a mesh while preserving its geometric shape.

3.1.1 Progressive Mesh Representation

According to Hoppe [29], an input 2-manifold triangle mesh $\widehat{\mathcal{M}}$ can be reduced via a series of $N_\Theta > 0$ edge collapses Θ into its minimal representation $\mathcal{M}_0$. that is:

$$\widehat{\mathcal{M}} = \mathcal{M}_{N_\Theta} \overset{\Theta_{N_\Theta - 1}}{\longmapsto} \mathcal{M}_{N_\Theta - 1} \overset{\Theta_{N_\Theta - 2}}{\longmapsto} \dots \overset{\Theta_1}{\longmapsto} \mathcal{M}_1 \overset{\Theta_0}{\longmapsto} \mathcal{M}_0. \tag{3.1}$$

Sequence $\{\Theta_i\}_{i=0}^{N_\Theta - 1}$ of edge collapses needs to be chosen so that the quality of the approximation $\mathcal{M}_i,\ i \in \{0, ..., N_\Theta - 1\}$ is maximized.

Definition 3.1.1. (Progressive Mesh) An ordered pair $(\mathcal{M}_0, \{\Theta_i^{-1}\}_{i=0}^{N_\Theta - 1})$ is called a *progressive mesh*.

Hence, all information needed to produce any mesh in the LOD sequence (3.1) is stored in the simplest base mesh, and a log of edge splits $\Theta_i^{-1},\ i \in \{0, ..., N_\Theta - 1\}$ (see Fig. 3.1).

Moreover, we can also interpolate between meshes $\mathcal{M}_{i+1}$ and $\mathcal{M}_i$ which only differ by a single tessellation-changing operation Θ_i^{-1} (see Definition 1.1.27). The interpolated mesh $\mathcal{M}_{i+1}^{\lambda}$ parametrized by $\lambda \in [0, 1)$ has the same combinatorial structure as mesh $\mathcal{M}_{i+1}$ from the previous step in simplification. These meshes, referred to as *geomorphs*, can be used for continuous transition between different LODs for a progressive mesh. In fact, geomorphs can be constructed between any two meshes in the progressive mesh representation. This is possible because for $l \in \{1, ..., N_\Theta\}$ a composition of edge collapses $A^\Theta = \Theta_{i_1} \circ ... \circ \Theta_{i_l}$ is surjective [29].

3.1.2 Optimization With Progressive Meshes

The goal of Hoppe et al. [32] was to find a mesh $\mathcal{M}$ that provides a good fit for a set of points $\{\mathbf{x}_1, ..., \mathbf{x}_M\} \subset \mathbb{E}^3$ with a small-enough number of vertices. This reduces to an optimization problem for an energy function:

$$E(\mathcal{M}) = E_{\text{dist}}(\mathcal{M}) + E_{\text{rep}}(\mathcal{M}) + E_{\text{spring}}(\mathcal{M}), \tag{3.2}$$

with *distance* and *representation energies*:

$$E_{\text{dist}}(\mathcal{M}) = \sum_{i=1}^{M} d^2(\mathbf{x}_i, \overline{\mathcal{M}}), \quad E_{\text{rep}}(\mathcal{M}) = c_{\text{rep}} M,\ c_{\text{rep}} > 0,$$

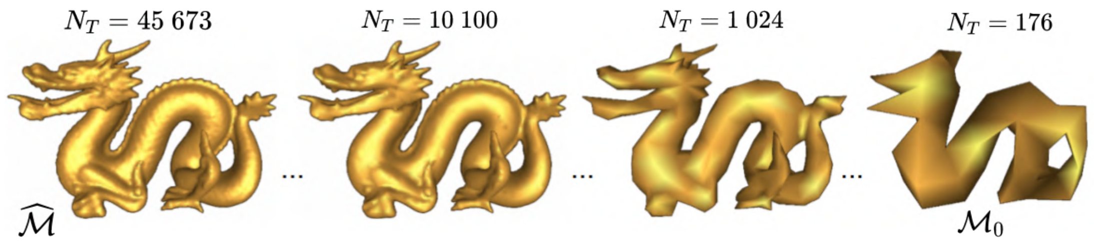


Figure 3.1: A sequence of progressive meshes of the *Stanford Dragon* for different triangle counts N_T. Source: lecture slides [31].

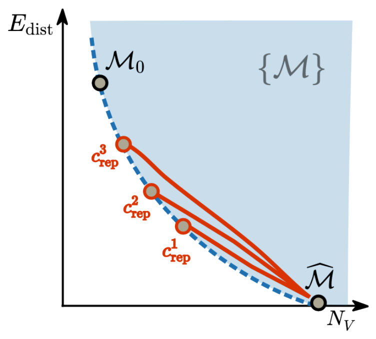


Figure 3.2: The space of meshes $\{\mathcal{M}\}$ illustrated as the shaded region in the (N_V, E_{dist})-plane where N_V is the vertex count. The solid lines represent optimization paths for different values of representation weight c_{rep}.

where $d(\mathbf{x}, \overline{\mathcal{M}}) = \inf_{\mathbf{p}\in\overline{\mathcal{M}}}\{\|\mathbf{x}-\mathbf{p}\|\}$ is the Euclidean distance to mesh $\mathcal{M}$, and *spring energy* for each edge:

$$E_{\text{spring}}(\mathcal{M}) = \sum_{e\in\mathcal{M}} \kappa\|\mathbf{v}_e^{(0)} - \mathbf{v}_e^{(1)}\|^2.$$

Point set $\{\mathbf{x}_1, ..., \mathbf{x}_M\}$ is chosen on the input mesh $\widehat{\mathcal{M}}$ randomly over its vertices as well as on faces.

The overall energy function (3.2) is minimized over the space of all meshes $\{\mathcal{M}\}$ available for starting mesh $\widehat{\mathcal{M}}$ via edge collapses Θ (see Fig. 3.2). Optimization, described in [32], consists of two loops (outer and inner). The outer loop optimizes over the combinatorial data choosing from a set of *legal moves* which represent operations Θ, Θ^{-1}, and an edge flip Φ included to allow the optimization to "tunnel" through small peaks in energy function (3.2). The inner loop, on the other hand, optimizes vertex positions. For performance reasons, the inner loop optimizes only one vertex position $\mathbf{v}$, and considers only the effect of vertices in the neighborhood $\mathcal{N}(\mathbf{v})$. To avoid producing self-intersections, the maximum dihedral angle of edges in $\mathcal{N}(\mathbf{v})$ cannot exceed a given threshold [29].

3.1.3 View-Dependent Refinement

In [30], Hoppe presents a method for dynamically refining a simplified 3D mesh model based on the viewer's position which is further optimized in [33] by parallelization. The view-dependent refinement algorithm builds on the concept of progressive meshes (see Section 3.1.1) by allowing the mesh to be refined in regions that are close to the viewer, while maintaining a lower level of detail in more distant regions. This results in a more efficient representation of the mesh during rendering, allowing for faster rendering times and reduced memory usage.

According to [30] a loaded progressive mesh $(\mathcal{M}_0, \{\Theta_i^{-1}\}_{i=0}^{N_\Theta - 1})$ can be used to construct a *vertex hierarchy* (see Fig.3.3) with mesh $\mathcal{M}_0$ with the lowest LOD forming the sequence of root vertices for the forest of binary trees storing mesh vertices. The parent-child relation in the hierarchy corresponds to operation Θ^{-1} peformed on a vertex.

View-dependent refinement uses a specialized query function, `qrefine`, which filters vertices $\mathbf{v}$ that should not be split based on the current view according to the following criteria:

(1) $\mathbf{v}$ is outside of the view frustum.

(2) $\mathbf{v}$ belongs to an edge e in the *backfacing region* of $\mathcal{M}_i$.

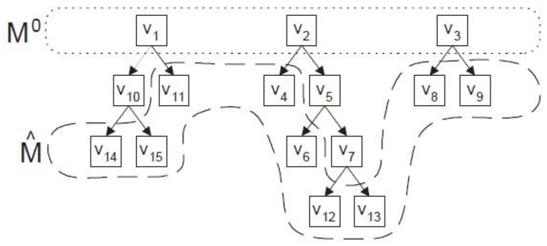


Figure 3.3: The vertex hierarchy forming a forest in which the roots of the binary trees are the vertices of the coarsest mesh $\mathcal{M}_0$. The leaves, on the other hand, represent the original mesh $\widehat{\mathcal{M}}$ with the highest LOD. Source: Hoppe [30].

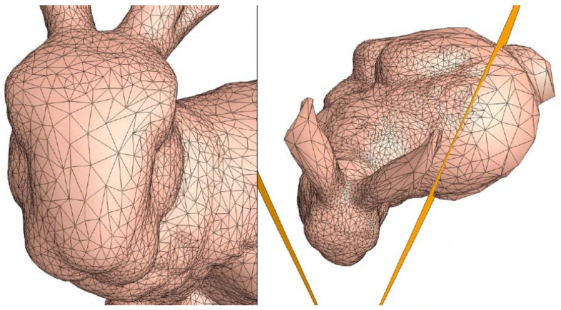


Figure 3.4: View-dependent refinement of the *Stanford Bunny* mesh, tessellated according to the view frustum shown as orange slicing planes in the left view. Source: Hoppe [30].

(3) The *screen space error* of $\mathbf{v}$ is less than some tolerance.

The parameters for the computation of screen space error are computed for each vertex during the construction of the progressive mesh representation. Since the filtering function is evaluated many times per frame, it needs to be fast.

Resulting selectively-refined mesh $\mathcal{M}^S$ is then chosen as a view-dependent cut of the vertex hierarchy by using the three filtering criteria. If $\mathcal{M}^A$ and $\mathcal{M}^B$ are two distinct selectively-refined meshes (for two different view frusta, for example), we can geomorph between them in $O(N_V^A + N_V^B)$ time in the worst case, where N_V^A and N_V^B are the vertex counts of the two meshes [29].

3.2 Mesh Compression

Progressive meshes $(\mathcal{M}_0, \{\Theta_i^{-1}\}_{i=0}^{N_\Theta - 1})$ (see Definition 3.1.1), introduced by Hoppe [29], have lower memory requirements than that of the typical input $\widehat{\mathcal{M}}$. The locations of the vertex split operations Θ_i^{-1} can be encoded more concisely than by storing all three indices of vertices $\mathbf{v}_e^*, \mathbf{v}^L$, and $\mathbf{v}^R$, and since Θ_i^{-1} has a local effect, significant coherence in position data can be expected. For example, since the positions of $\mathbf{v}_e^{(0)}$, and $\mathbf{v}_e^{(1)}$ after splitting $\mathbf{v}_e^*$ can be predicted, and do not need to be stored.

The above approach provides high compression ratios with minimal loss of visual quality, making it well-suited for applications that require efficient storage and transmission of 3D mesh data. The use of a simplified sequence of meshes also makes it possible to render the mesh at different levels of detail, depending on the available memory capacity.

As the size of the input mesh $\widehat{\mathcal{M}}$ grows, however, the number of vertices in the vertex hierarchy also increases, making it more difficult to store and transmit the compressed mesh efficiently. This can result in longer compression and decompression times, as well as increased memory usage. To address this issue, researchers have proposed various extensions and modifications to the progressive mesh data structure.

In their 1999 paper, Cohen et al. [19] proposed a new algorithm for compressing progressive meshes (PMs) that addresses some of the limitations of Hoppe's original PM algorithm, including the storage size of the PM data structure. The main idea was to use *triangle strips* to further reduce the storage size of the data structure.

Alliez and Desbrun [4] claim that removing a vertex of valence more than six can increase entropy and result in a lower compression rate. Thus, this strategy is not ideal for a compression algorithm that aims to minimize bit cost. They also introduced an inverse $\sqrt{3}$ simplification process that maintained valence regularity during progressive encoding.

3.3 The Cutting Edge: Virtualized Geometry Pipeline

View-dependent geometry optimization has seen significant advancements in recent years, and one of the most notable developments in this area is the *Virtualized Geometry* pipeline. This cutting-edge technology is a key feature of the Nanite™system, which is a revolutionary real-time rendering technology introduced in Unreal Engine 5™.

The Virtualized Geometry pipeline leverages hardware-accelerated virtual texturing and sparse virtual texturing techniques to render large and complex scenes with extremely high levels of detail. This is achieved by dynamically streaming in and out portions of the scene's geometry as needed, based on the viewer's position and other factors such as object occlusion and LOD management. The pipeline allows for the rendering of millions of triangles per frame, while minimizing memory usage and improving performance compared to traditional geometry processing techniques. This new technology has the potential to revolutionize the way 3D graphics and visualization are implemented, allowing for the creation of more detailed and immersive virtual environments than ever before.

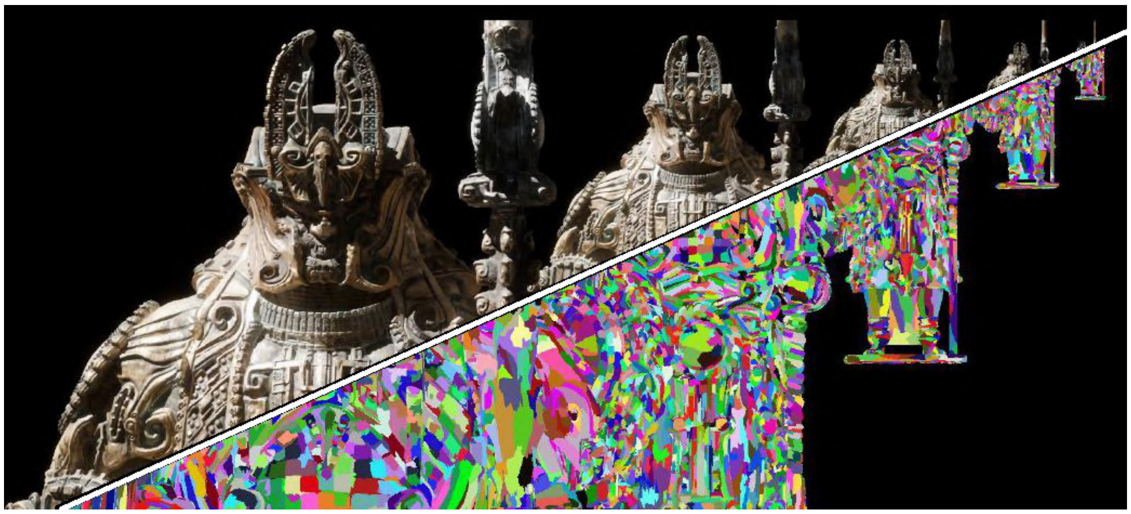


Figure 3.5: A visualization of micropolygon clusters in the Nanite pipeline. Source: Brian Karis [41]

At its core, Nanite™uses a virtualized *micropolygon* geometry representation [15], where the mesh is divided into small, fixed-size triangles called micropolygons. These micropolygons are then stored in a *sparse voxel octree* (SVO) structure [45]. Organized into *clusters* (see Fig. 3.5), the micropolygons can then be used for culling. During rendering, the SVO is traversed and only the visible micropolygons are rasterized, resulting in high performance and efficient memory usage.

In addition to micropolygon rendering, Nanite™also supports various advanced features such as dynamic global illumination and volumetric fog. The pipeline is optimized for modern hardware architectures, including multi-core CPUs and GPUs with hardware-accelerated ray tracing capabilities. Overall, Nanite™represents a significant step forward in real-time rendering, enabling developers to create more immersive and detailed scenes than ever before.

Chapter 4

Project of Dissertation

The final chapter focuses on our research goals for the remainder of the PhD study. We plan to simplify meshes using shrink-wrapping to load large files, and explore texture data and volumetric representations. The chapter is divided into three sections, each addressing a potential research direction.

4.1 Previewing Large Mesh Files

Despite the extent of advancements from the hardware side, large polygonal datasets, with representations exceeding memory capacity of the application process, still pose a problem. Although the working memory can be extended in many applications, such as MeshLab™, the performance of the rendering pipeline itself is often not optimized for gigabytes of input data. The datasets themselves, on the other hand, only seem to grow in size due to modern high-resolution photogrammetry and 3D scanning.

Let $\mathcal{M}_{\max}$ be the input mesh with a very large amount of vertices and connectivity data stored in a file. We propose the following approach:

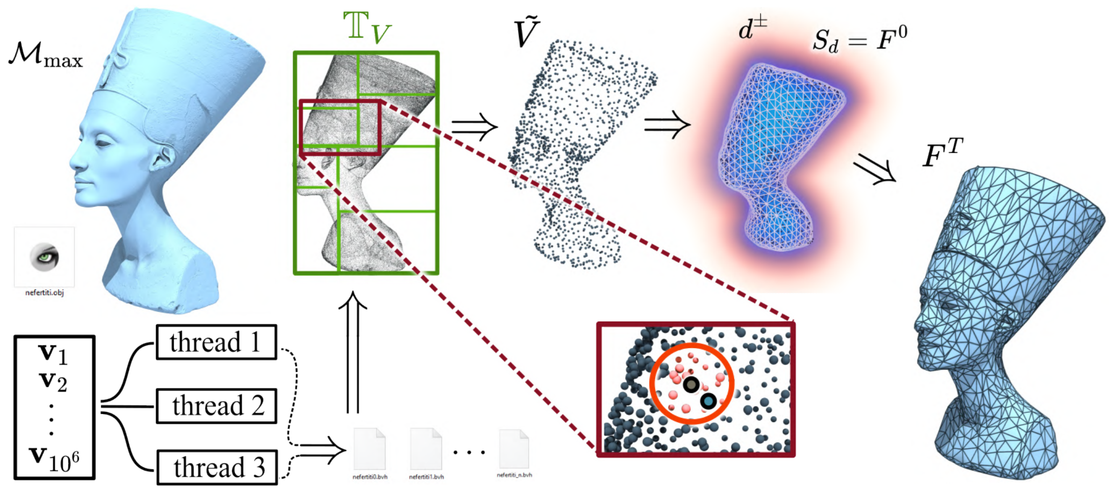


Figure 4.1: Proposal for the pipeline for the simplification of large mesh files.

(1) Read the vertex buffer V of the input data in parallel, using for example `mmap`[1], and stream it as point cloud data into the next stage.

(2) Construct a kD-tree $\mathbb{T}_V$ for the point cloud data V, and select a subset $\tilde{V} \subset V$ with much smaller cardinality than the original data using the kD-tree's *radius search query*. Namely, for some point $\mathbf{v} \in V$ we randomly choose the next vertex within radius $r > 0$ from $\mathbf{v}$.

(3) The reduced point cloud $\tilde{V}$ is then either triangulated using a Delaunay-based technique [8]. Given vertex normal data, we can use *Poisson reconstruction* [42]. If the input contains a lot of internal structure we may also use the shrink-wrapping approach from Chapter 2.

(4) The triangulated reduced geometry will then be optimized according to the view frustum (see Section 3.1.3).

This pipeline can then be implemented as an extension to MeshLab (see Fig. 4.1). The simplified mesh can then be rendered in the application viewport and stored as backup on hard drive. Moreover, the complex files can be stored at different LODs on disk, including the entire vertex hierarchy forest (see Section 3.1.1).

We intend to test this approach for meshes with 50M triangles or more, including some procedurally generated fractal geometries. It is also important to leverage the initial preprocessing stage with multithreading, while also informing the user about the progress of this process.

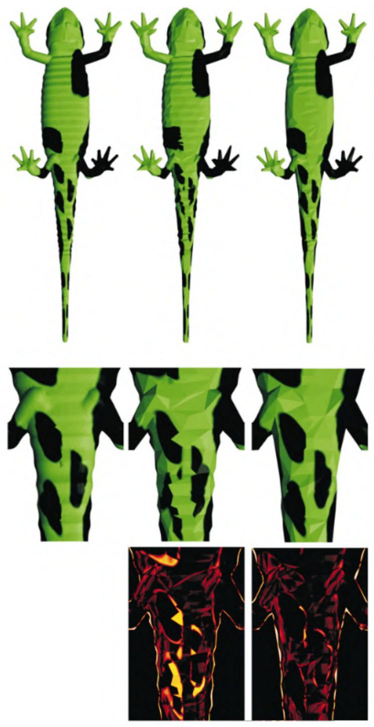


Figure 4.2: Comparison of the original textured mesh (left), geometry-based (middle), and image-based simplification (right) with displacement heatmaps. Image credit: Lindstrom et al. [49].

4.2 Refinement Based on Texture Data

An additional technique which may be integrated into the processing pipeline for large meshes is the utilization of a subset of the mesh space $\{\mathcal{M}\}$ which incorporate texture data according to perceptual criteria. If $I : \overline{\mathcal{M}} \to [0,1]^4$ is a texture defined on the mesh surface, there could be varying criteria for the quality of triangulation based on different color channels.

From the work of Hoppe on progressive meshes (see Section 3.1.1) other researchers have explored various extensions and improvements to the progressive mesh framework, including those that incorporate texture data into the refinement process.

Lindstrom et al. proposed a method for simplification based on multiple views of a textured mesh [49]. Late Sander et al. provided a framework for properly mapping texture data onto Hoppe's progressive meshes [61]. The latest advances in differentiable rendering [27], on the other hand push this development to the limit.

4.3 Using FReps for Scene Optimization

The conversion of the entire scene with high-poly meshes to an entirely volumetric form is perhaps the hardest challenge of the ones stated in this chapter. The process of voxelization, which involves converting a mesh into a volumetric representation, is computationally intensive

[1]A C function for mapping between address space of a process and a given file.

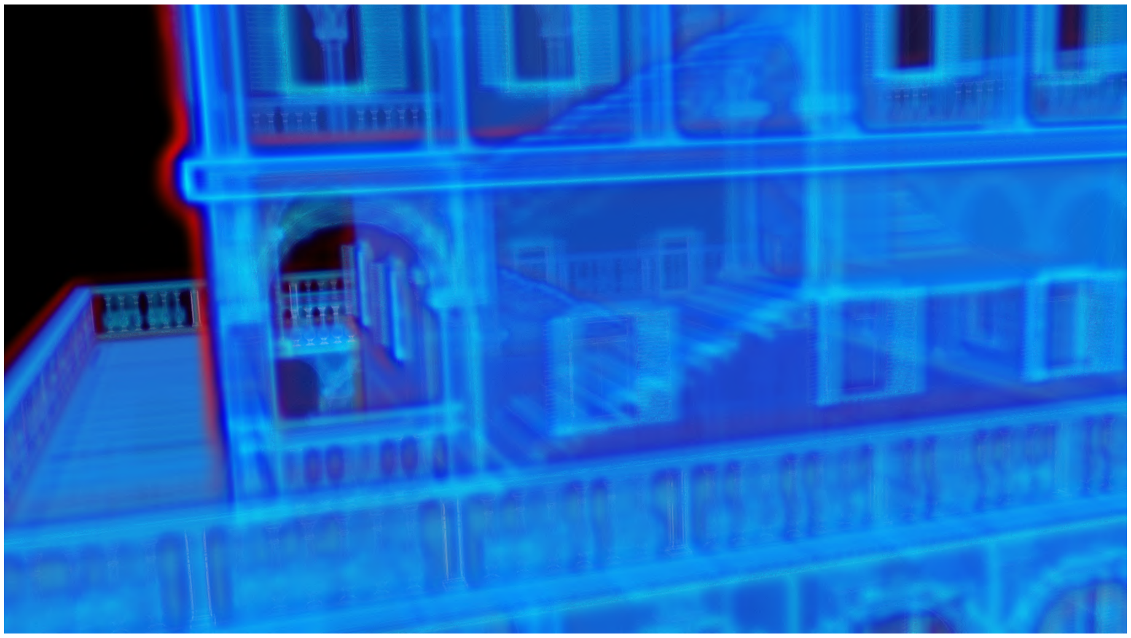


Figure 4.3: The distance field to a free *palace* mesh [75] with 1M triangles and 500K vertices on a grid $\overline{G}$ with resolution 600^3 completed in about 21 seconds on AMD Ryzen 7. The exported voxel `.vti` file (ASCII) has 1.1 GB.

and requires a significant amount of memory (see Fig. 4.3). Fitting a set of basis functions onto a dataset of mesh vertices while maintaining a high level of detail [76, 47] is another challenge we face. Both performance and memory optimization will be crucial in tackling this challenge. Therefore, finding a balance between maintaining a high level of detail and optimizing performance and memory usage will be a significant hurdle in our research.

Firstly, we will need to leverage the bottlenecks for preprocessing speed with high-performing solvers for basis function fitting. This will require us to identify and leverage bottlenecks in the process and fine-tune our approach to ensure the most efficient use of computational resources.

In addition, it is crucial that we find a way to maintain a sufficient amount of detail within surface meshes while still fitting them to our volumetric representation. Preserving details within the mesh is important, as it can greatly affect the quality of the final result. We will need to explore different approaches to ensure that the mesh retains its integrity while still being converted to a volumetric form.

Overall, our research will require a balance between preprocessing speed, high-performing solvers, and mesh detail preservation. Finding an effective and efficient way to manage these factors will be essential to achieving our goals.

References

[1] 3D Systems, Inc. Stereolithography interface specification, 1988.

[2] 3mf Consortium. 3mf volumetric extension. https://github.com/3MFConsortium/spec_volumetric, 2021.

[3] Alarcão, D., Bento Coelho, J., Camilo, T., and Tenenbaum, R. On the use of hybrid methods for fast acoustical simulations in enclosures. In *Proceedings of Forum Acusticum* (2002), pp. 1–7.

[4] Alliez, P., and Desbrun, M. Progressive compression for lossless transmission of triangle meshes. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), pp. 195–202.

[5] Čavarga, M. Advection-driven shrink-wrapping of triangulated surfaces. https://cescg.org/cescg_submission/advection-driven-shrink-wrapping-of-triangulated-surfaces/, 2022.

[6] Čavarga, M. Implicit surface wrap. https://github.com/MCInversion/ImplicitSurfaceWrap, 2022.

[7] Bern, M., Eppstein, D., and Erickson, J. Flipping cubical meshes. *Engineering with Computers 18* (08 2001), 1–20.

[8] Bernardini, F., Mittleman, J., Rushmeier, H., Silva, C., and Taubin, G. The ball-pivoting algorithm for surface reconstruction. *IEEE transactions on visualization and computer graphics 5*, 4 (1999), 349–359.

[9] Bloomenthal, J. Polygonization of implicit surfaces. *Computer Aided Geometric Design 5*, 4 (1988), 341–355.

[10] Botsch, M., and Kobbelt, L. A remeshing approach to multiresolution modeling. pp. 189–96.

[11] Botsch, M., Kobbelt, L., Pauly, M., Alliez, P., and Lévy, B. *Polygon Mesh Processing.* CRC press, 2010.

[12] Botsch, M., Steinberg, S., Bischoff, S., Kobbelt, L., and Aachen, R. Openmesh - a generic and efficient polygon mesh data structure. 1–5.

[13] Bourke, P. Polygonising a scalar field. http://paulbourke.net/geometry/polygonise/, 1994.

[14] Bærentzen, A., and Aanæs, H. Signed distance computation using the angle weighted pseudonormal. *IEEE transactions on visualization and computer graphics 11* (06 2005), 243–53.

[15] Brunhaver, J. S., Fatahalian, K., and Hanrahan, P. Hardware implementation of micropolygon rasterization with motion and defocus blur. In *High Performance Graphics* (2010), pp. 1–9.

[16] Campagna, S., Kobbelt, L., and Seidel, H.-P. Directed edges—a scalable representation for triangle meshes. *Journal of Graphics Tools 3*, 4 (1998), 1–11.

[17] Catmull, E., and Clark, J. Recursively generated b-spline surfaces on arbitrary topological meshes. 350–355.

[18] Cheng, S.-W., and Jin, J. Edge flips in surface meshes. *Discrete & Computational Geometry 54*, 1 (2015), 110–151.

[19] Cohen-Or, D., Levin, D., and Remez, O. Progressive compression of arbitrary triangular meshes. In *IEEE visualization* (1999), vol. 99, pp. 67–72.

[20] Daniel, P., Medl'a, M., Mikula, K., and Remešíková, M. Reconstruction of surfaces from point clouds using a lagrangian surface evolution model. pp. 589–600.

[21] de Berg, M., Kreveld, M. V., Overmars, M., and Cheong, O. *Computational Geometry: Algorithms and Applications.* Springer, 2000.

[22] DeRose, T., Kass, M., and Truong, T. Subdivision surfaces in character animation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques* (1998), pp. 85–94.

[23] Dunyach, M., Vanderhaeghe, D., Barthe, L., and Botsch, M. Adaptive remeshing for real-time mesh deformation. In *Eurographics 2013 - Short Papers* (2013), p. 29–32.

[24] Garland, M., and Heckbert, P. S. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (1997), pp. 209–216.

[25] Geiss, R. Metaballs (also known as: Blobs). `http://www.geisswerks.com/ryan/BLOBS/blobs.html`, 2000.

[26] Hart, J. C. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer 12*, 10 (1996), 527–545.

[27] Hasselgren, J., Munkberg, J., Lehtinen, J., Aittala, M., and Laine, S. Appearance-driven automatic 3d model simplification. In *EGSR (DL)* (2021), pp. 85–97.

[28] Hilton, A., Stoddart, A. J., Illingworth, J., and Windeatt, T. Marching triangles: range image fusion for complex object modelling. In *Proceedings of 3rd IEEE international conference on image processing* (1996), vol. 2, IEEE, pp. 381–384.

[29] Hoppe, H. Progressive meshes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (1996), pp. 99–108.

[30] Hoppe, H. View-dependent refinement of progressive meshes. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (1997), pp. 189–198.

[31] Hoppe, H. Geometry processing algorithms. `http://graphics.stanford.edu/courses/cs468-10-fall/LectureSlides/09_Progressive_Meshes.pdf`, 2022.

[32] Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., and Stuetzle, W. Mesh optimization. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1993), SIGGRAPH '93, Association for Computing Machinery, p. 19–26.

[33] Hu, L., Sander, P. V., and Hoppe, H. Parallel view-dependent refinement of progressive meshes. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games* (2009), pp. 169–176.

[34] Hunt, W., Mark, W. R., and Stoll, G. Fast kd-tree construction with an adaptive error-bounded heuristic. In *2006 IEEE Symposium on Interactive Ray Tracing* (2006), pp. 81–88.

[35] Hurtado, F., Noy, M., and Urrutia, J. Flipping edges in triangulations. *Discrete & Computational Geometry 22* (1996), 333–346.

[36] Hurtado, J., Montenegro, A., Gattass, M., Carvalho, F., and Raposo, A. Enveloping cad models for visualization and interaction in xr applications. *Engineering with Computers* (2022), 1–19.

[37] Huska, M., Medl'a, M., Mikula, K., and Morigi, S. Lagrangian evolution approach to surface-patch quadrangulation. *Applications of Mathematics 66* (03 2021), 1–43.

[38] Italian National Research Council Institute (ISTI). The visualization and computer graphics library (vcglib). `https://github.com/cnr-isti-vclab/vcglib`, 2016.

[39] Jones, M. W., Baerentzen, J. A., and Sramek, M. 3d distance fields: A survey of techniques and applications. *IEEE Transactions on visualization and Computer Graphics 12*, 4 (2006), 581–599.

[40] Ju, T., Losasso, F., Schaefer, S., and Warren, J. Dual contouring of hermite data. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (2002), pp. 339–346.

[41] Karis, B., Stubbe, R., and Wihlidal, G. Nanite—a deep dive. In *ACM SIGGRAPH* (2021), vol. 21.

[42] Kazhdan, M., Bolitho, M., and Hoppe, H. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing* (2006), vol. 7, p. 0.

[43] Knupp, P. M. Algebraic mesh quality metrics for unstructured initial meshes. *Finite Elem. Anal. Des. 39*, 3 (jan 2003), 217–241.

[44] Kobbelt, L., Vorsatz, J., Labsik, U., and Seidel, H.-P. A shrink wrapping approach to remeshing polygonal surfaces. *Computer Graphics Forum 18* (04 2000), 119–130.

[45] Laine, S., and Karras, T. Efficient sparse voxel octrees. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games* (2010), pp. 55–63.

[46] Lawson, C. L. Transforming triangulations. *Discrete Mathematics 3*, 4 (1972), 365–372.

[47] Li, Q., Wills, D., Phillips, R., Viant, W. J., Griffiths, J. G., and Ward, J. Implicit fitting using radial basis functions with ellipsoid constraint. In *Computer Graphics Forum* (2004), vol. 23, Wiley Online Library, pp. 55–69.

[48] Liepa, P. Filling holes in meshes. pp. 200–205.

[49] Lindstrom, P., and Turk, G. Image-driven simplification. *ACM Transactions on Graphics (ToG) 19*, 3 (2000), 204–241.

[50] Loop, C. *Smooth Subdivision Surfaces Based on Triangles*. PhD thesis, January 1987.

[51] Lorensen, W. E., and Cline, H. E. Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics 21*, 4 (1987), 163–169.

[52] Meyer, M., Desbrun, M., Schröder, P., and Barr, A. Discrete differential-geometry operators for triangulated 2-manifolds. p. 37–57.

[53] Mikula, K., Ševčovič, D., and Balažovjech, M. A simple, fast and stabilized flowing finite volume method for solving general curve evolution equations. *Communications in Computational Physics 7* (11 2010), 195–211.

[54] Mikula, K., Remešíková, M., Sarkoci, P., and Ševčovič, D. Manifold evolution with tangential redistribution of points. *SIAM Journal on Scientific Computing 36* (07 2014), A1384–A1414.

[55] Moulaeifard, M., Wellmann, F., Bernard, S., de la Varga, M., and Bommes, D. Subdivide and conquer: Adapting non-manifold subdivision surfaces to surface-based representation and reconstruction of complex geological structures. *Mathematical Geosciences* (09 2022).

[56] Osthoff, C., Souto, R. P., Vilasbôas, F., Grunmann, P., Dias, P. L. S., Boito, F., Kassick, R., Pilla, L., Navaux, P., Schepke, C., et al. Improving atmospheric model performance on a multi-core cluster system. *Atmospheric Model Applications* (2012), 1–25.

[57] Pasko, A., Adzhiev, V., Sourin, A., and Savchenko, V. Function representation in geometric modeling: concepts, implementation and applications. *The visual computer 11* (1995), 429–446.

[58] Quilez, I. Distance functions. `https://iquilezles.org/articles/distfunctions/`, 2018.

[59] Rashid, T., Sultana, S., and Audette, M. A. Watertight and 2-manifold surface meshes using dual contouring with tetrahedral decomposition of grid cubes. *Procedia engineering 163* (2016), 136–148.

[60] Sanchez, M., Fryazinov, O., and Pasko, A. Efficient evaluation of continuous signed distance to a polygonal mesh. In *Proceedings of the 28th Spring Conference on Computer Graphics* (2012), pp. 101–108.

[61] Sander, P. V., Snyder, J., Gortler, S. J., and Hoppe, H. Texture mapping progressive meshes. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), pp. 409–416.

[62] Sandia National Laboratories. Metrics for triangular elements. `https://www.sandia.gov/files/cubit/15.3/help_manual/WebHelp/mesh_generation/mesh_quality_assessment/triangular_metrics.htm`, 2017.

[63] Schaefer, S., Ju, T., and Warren, J. Manifold dual contouring. *IEEE Transactions on Visualization and Computer Graphics 13*, 3 (2007), 610–619.

[64] Sharp, N., and Crane, K. You can find geodesic paths in triangle meshes by just flipping edges. *ACM Trans. Graph. 39*, 6 (nov 2020), 1–15.

[65] Sharp, N., Crane, K., et al. geometry-central. `www.geometry-central.net`, 2019.

[66] Shewchuk, J. R. What is a good linear finite element? interpolation, conditioning, anisotropy, and quality measures.

[67] Shewchuk, J. R. Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. In *Applied Computational Geometry Towards Geometric Engineering: FCRC'96 Workshop, WACG'96 Philadelphia, PA, May 27–28, 1996 Selected Papers* (2005), Springer, pp. 203–222.

[68] Sieger, D., and Botsch, M. Design, implementation, and evaluation of the surface_mesh data structure. In *Proceedings of the 20th International Meshing Roundtable* (2012), pp. 533–550.

[69] Sieger, D., and Botsch, M. The polygon mesh processing library. `http://www.pmp-library.org`, 2019.

[70] Smith, S. Fast robust automated brain extraction. *Human brain mapping 17* (2002), 143–55.

[71] Sustainability of Digital Formats. Stl (stereolithography) file format family. `https://www.loc.gov/preservation/digital/formats/fdd/fdd000504.shtml`, 2019.

[72] Sustainability of Digital Formats. Wavefront obj file format. `https://www.loc.gov/preservation/digital/formats/fdd/fdd000507.shtml`, 2019.

[73] Tereshin, A., Pasko, A., Fryazinov, O., and Adzhiev, V. Hybrid function representation for heterogeneous objects. *Graphical Models 114* (2021), 101098.

[74] The CGAL Project. The computational geometry algorithms library (cgal). `https://www.cgal.org`, 1995.

[75] User: 13560838706you. Royalty-free palace mesh. `https://www.cgtrader.com/free-3d-models/exterior/house/palace-6f6e2c29-c0bb-488b-8d4b-1c49755f2b7a`, 2022.

[76] Yngve, G., and Turk, G. Robust creation of implicit surfaces from polygonal meshes. *IEEE transactions on visualization and computer graphics 8*, 4 (2002), 346–359.

[77] Zhao, H. A fast sweeping method for eikonal equations. *Math. Comput. 74* (2005), 603–627.

[78] Zhong, D., Zhang, J., and Wang, L. Fast implicit surface reconstruction for the radial basis functions interpolant. *Applied Sciences 9*, 24 (2019), 5335.

[79] Zhou, Q., Grinspun, E., Zorin, D., and Jacobson, A. Mesh arrangements for solid geometry. *ACM Trans. Graph. 35*, 4 (jul 2016).